uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Electrónica Industrial y Automática
2018 - 2019

Trabajo Fin de Grado

"Control de un vehículo inteligente mediante aplicación móvil y Arduino"

Guillermo Fraga García-Yanes

Tutor

David Griol Barres

Leganés, Junio 2019

RESUMEN

En el presente Trabajo de Fin de Grado se ha desarrollado una aplicación para la plataforma Android en la que se pretende realizar un acercamiento a varios campos y conceptos.

En esta aplicación se realiza una conexión entre un dispositivo Android y un módulo Bluetooth que trabaja con una placa Arduino, donde se realizan diferentes acciones llevadas a cabo por el usuario; por ello se realiza un estudio exhaustivo acerca de estas dos plataformas y las herramientas que utilizan.

Dicha aplicación permite conocer más profundamente conceptos como el Internet de las Cosas o IoT (*Internet of Things*), mediante la conexión de un Smartphone con objetos de la vida cotidiana, también permite conocer las diferentes tecnologías de comunicación entre dispositivos y cuáles son las opciones para realizar una conexión remota entre ellos.

Para el presente proyecto se ha decidido que el objeto a interactuar con el Smartphone sea un vehículo inteligente, un prototipo a pequeña escala de lo que podría tratarse como el vehículo que se utiliza usualmente en la vida cotidiana, permitiendo así comprobar que la aplicación desarrollada funciona correctamente. Por ello, el proyecto también supone un pequeño acercamiento al campo de la electrónica y la programación, donde se busca interpretar los datos enviados y recibidos por el Smartphone y el módulo Bluetooth para tomar diferentes decisiones, como obtener la temperatura del interior de este, controlarlo o conocer su localización.

Palabras clave: Android, Arduino, tecnologías de comunicación, IoT, electrónica, vehículo inteligente.

AGRADECIMIENTOS

Por mi abuela, que ahora descansa en paz, mi familia y el apoyo incondicional que siempre he recibido de su parte; y mis amigos, que siempre han estado ahí cuando se les ha necesitado.
Agradecer también a mi Tutor, David Griol Barres, su tiempo y amabilidad para poder llevar a cabo este TFG.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN

En la introducción se ofrecen los diferentes motivos acerca de la elección del tema; el control de un vehículo inteligente con diferentes funciones como Trabajo de Fin de grado y los diferentes objetivos que se quieren alcanzar con dicho trabajo. Se analizarán los antecedentes históricos relacionados con la automatización y el control de los dispositivos electrónicos de manera remota para facilitar las tareas del ser humano, así como el amplio futuro que suponen estos campos en nuestra sociedad. También se detallarán los problemas iniciales que limitaban el desarrollo y comienzo del proyecto, se hará un estudio acerca de las herramientas utilizadas en el trabajo, como Arduino, el S.O. Android, la tecnología B*luetooth* y se mencionarán otras que han sido necesarias para el desarrollo de éste. Por último se analizará la estructura que se ha llevado a cabo para una mejor comprensión del lector.

1.1 Introducción

La elección de este Trabajo de Fin de Grado radica tanto en la ampliación de los conocimientos adquiridos a lo largo de la carrera como en la satisfacción personal de realizar un proyecto individual completamente ligado a la electrónica, la programación y otras áreas relacionadas. Consiste en la fabricación, programación y desarrollo de un vehículo inteligente que será capaz de realizar tareas programadas. El proyecto girará en torno al concepto del ***IoT***, comúnmente conocido como *El internet de las cosas*. Este concepto se aplica sobre múltiples objetos físicos, ya sean máquinas, vehículos, electrodomésticos o cualquier objeto de donde el ser humano pueda obtener información valiosa para tomar una posterior decisión. Lo que se necesita para llevar a cabo el concepto es, como se acaba de comentar, un objeto físico, un sensor que recoja información y una red donde se transmita ésta información y se almacene, posteriormente el desarrollador podrá tratar los datos con el fin que desee, pudiendo, o no, realizar alguna acción como consecuencia.

Con el desarrollo de la tecnología, el IoT ha ido mejorando en consecuencia; en la figura 1.1 se observa la evolución en las inversiones anuales monetarias de

este concepto en los diferentes sectores industriales de Estados Unidos mediante un diagrama de barras verticales.

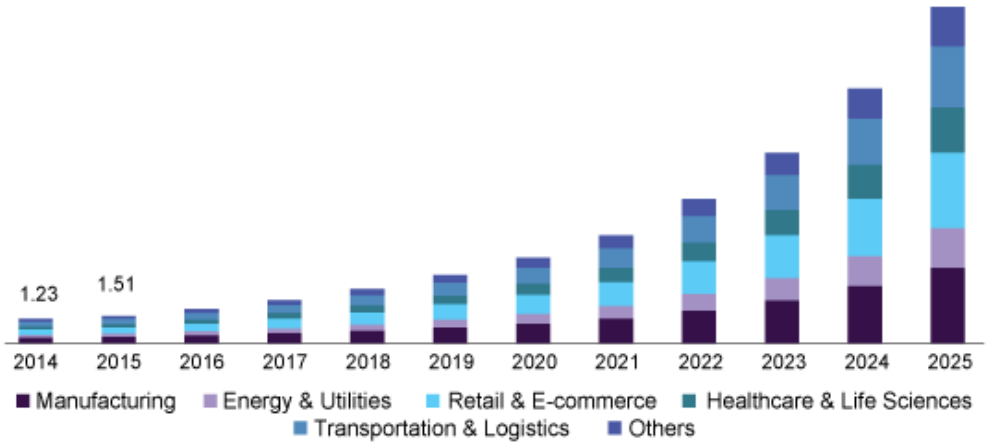


Figura 1.1 Inversiones anuales sobre el IoT en los diferentes sectores [1]

En 2014 se invirtieron 1,23 mil millones de dólares en total en esta tecnología, y se prevé que las cifras aumenten exponencialmente con el paso de los años.

Según las estadísticas y los datos recopilados acerca del avance tecnológico, las necesidades generadas y otros factores, se ha realizado una previsión sobre las inversiones que se realizarán en los próximos años.

La mayor focalización en 2025 se encontrará en el comercio electrónico y la venta *Retail* [2], es decir, en la venta al por menor, a múltiples clientes a partir de un *stock* masivo. Posteriormente se encontrará el sector de la producción y el sector del transporte y logística.

Todas las inversiones e investigaciones realizadas se harán con un fin y estarán dirigidas a diferentes sectores. En la vida cotidiana, las aplicaciones e intereses, de momento, tienen una distribución diferente a la que se encuentra en la figura anterior. En la figura 1.2 se encuentra un diagrama de barras horizontales proporcionada por *IoT Analytics*, donde se muestran los porcentajes en cuanto al interés de los usuarios por los diferentes departamentos.

La puntuación más alta recibe un 100% de interés y los demás porcentajes se evalúan en relación a dicho valor. La puntuación, como se puede observar,

deriva de tres fuentes; las búsquedas que se realizan en *Google*, lo que se comenta en la red social *Twitter* y lo que se escribe en *LinkedIn*.

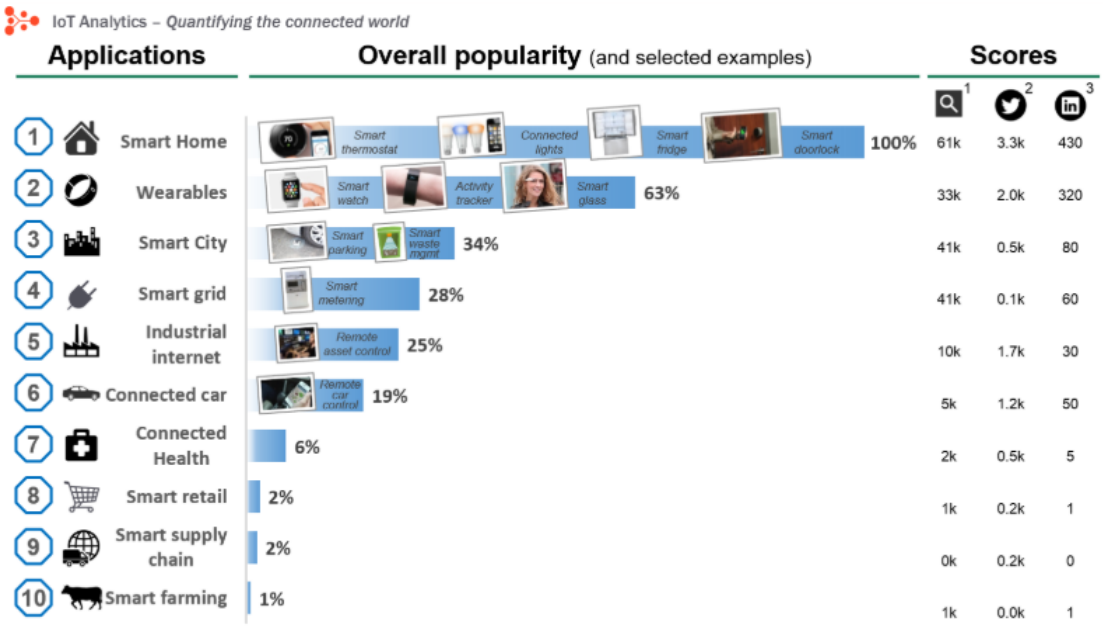


Figura 1.2 Intereses en 2015 en tecnologías IoT [3]

La mayoría de intereses se encuentran en la domótica de casa, nuestro lugar de descanso. En este, las personas quieren sentirse lo mas cómodas posibles, por lo que buscan automatizar la mayoría de dispositivos que se encuentran en ella. Se puede observar también como la tecnología corporal o "*Wearables"* también se encuentra en los objetivos de la sociedad, que incluye gafas inteligentes, industria textil y *Smartwatches* entre otros accesorios.

La tecnología aplicada en la agricultura o *"Smart farming"* ha cobrado más importancia en los últimos años, debido a que se ha abierto la posibilidad de trabajar con drones y máquinas inteligentes que trabajan de manera mucho mas rapida y eficaz.

En este proyecto se busca aplicar esta tecnología del IoT a los vehículos ya que es un campo algo inexplorado y realmente importante en la sociedad actual:

Para visualizar la importancia acerca de estos se utilizarán las encuestas realizadas por la *NTS* (*National Travel Survey* o Encuesta nacional de viajes) de Reino Unido acerca de los viajes que se efectúan en dicho país.

En 2017, según la NTS, se realizaron un promedio de 594 viajes por persona al año y recorrieron alrededor de 5104 millas (8214 km) en dicho año cada una.

En la siguiente figura se muestra de manera gráfica la distancia promedio recorrida, en millas, por persona, género y edad:

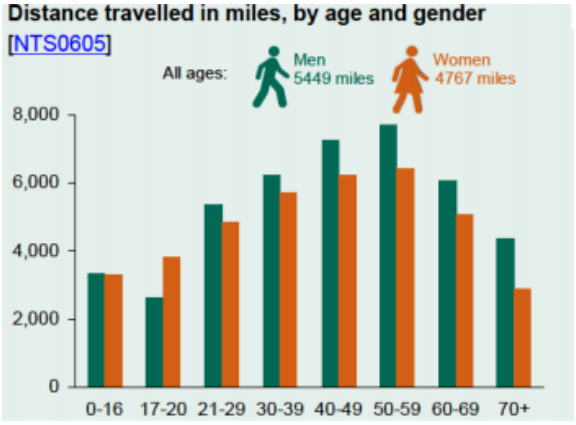


Figura 1.3 Distancia recorrida en coche en 2017 por edad y género [4]

Basándonos en estas estadísticas se concluye que gran parte de nuestro tiempo, medido de manera anual, lo invertimos en transporte a través del automóvil, ya que, el tiempo promedio de duración de un trayecto resulta en 22 minutos. Si este tiempo se cumple tanto en la ida como en la vuelta, al año podríamos haber pasado más de 250 horas al volante. La importancia de este campo radica en nuestra propia seguridad, ya que, según un informe realizado por las empresas RACE, BP y CASTROL en España a 37.000 conductores, el 58,76% de ellos admitía haber utilizado el móvil alguna vez mientras conducía. Si llevamos estas cifras fuera de estas encuestas resulta que 13.274.191 conductores han utilizado el móvil alguna vez [5], y esto supone un peligro para la seguridad de las personas.

En la siguiente figura se muestra un gráfico en el que se mide el uso que se le da al telefono móvil mientras se conduce:

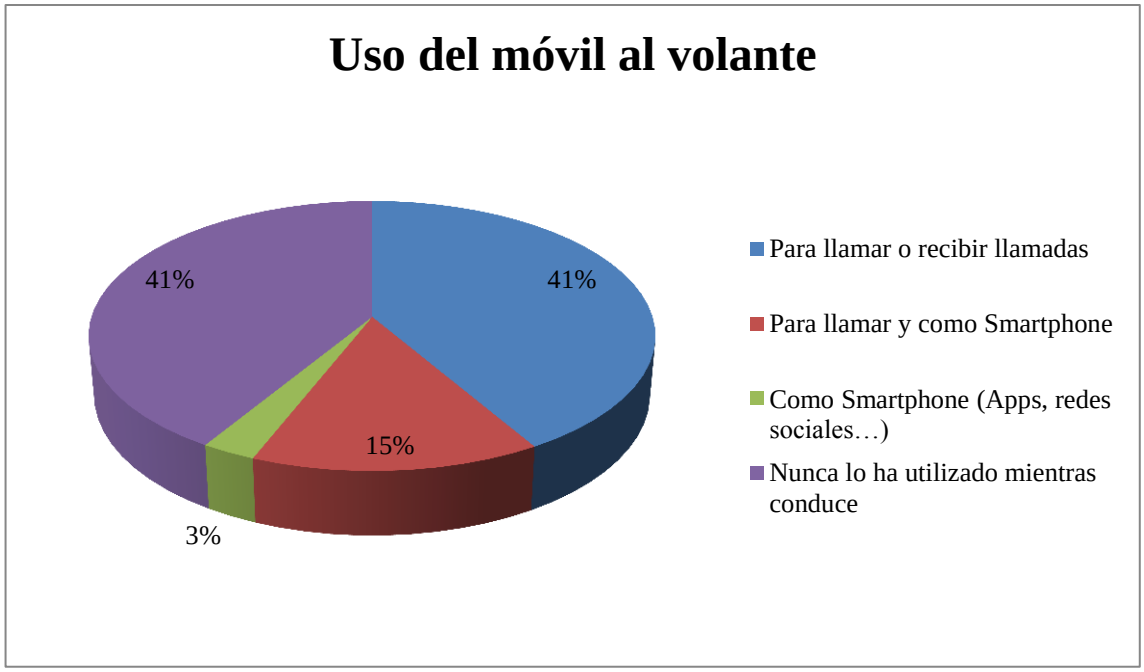


Figura 1.4 Uso del móvil al volante

Con estos datos se observa como la mayoría de personas que utilizan el móvil conduciendo lo hacen para atender llamadas, y un 21,34 % lo hace sin manos libres, un 15 como llamadas y Smartphone y un 3% solo como Smartphone.

Con esta información se puede concluir que si se aplican mejoras tecnológicas en los Smartphones, ya sea para realizar todas las acciones mediante un control por voz u otras funciones inteligentes que relacionen vehículo y Smartphone, el peligro que supone el uso de este al volante se reduciría drásticamente y con ello la seguridad de los demás conductores y peatones.

1.2 Historia y antecedentes

Desde que el ser humano hace uso de razón, continuamente se ha visto involucrado en la búsqueda de herramientas que facilitaran, reduciendo tiempos o esfuerzos, la vida de este.

A continuación, se hará un breve repaso de la historia y los antecedentes que preceden al tema en concreto.

1.2.1 Vehículos autónomos inteligentes

Este trabajo supone un pequeño acercamiento al futuro y presente de los **vehículos inteligentes** y su control. Si se mira hacia atrás en el tiempo, se puede observar como desde 1939 [6] surgieron vehículos que ya contaban con inteligencia artificial. A lo largo de los años ha ido avanzando la tecnología en gran medida, siendo cada vez la electrónica y la mecánica más sofisticada, eficiente y reducida en tamaño. Esto se puede observar en la propia historia, en 1994 surgen vehículos capaces de recorrer de manera autónoma más de mil kilómetros sin casi intervenciones humanas, como es el caso de los robots *VaMP* (Figura 1.5) y *Vita-2* por parte de *Daimler-Benz*, la actual *Mercedes Benz*, y *Dickmanns*, uno de los pioneros en visión por computador aplicado en vehículos.

Figura 1.5 "VaMP", el vehículo inteligente de 1994 [7]

Posteriormente se han conseguido verdaderos récords en el ámbito de conducción autónoma.

Pero la inteligencia artificial no solo se ha utilizado para esto; con el avance de la tecnología también se ha utilizado para conocer al conductor y tomar medidas preventivas o simplemente confortantes para este y su conducción. Ya están en marcha sistemas de detección de fatiga o sueño para evitar posibles accidentes, sistemas que según el uso de los intermitentes, la ausencia de movimiento en el volante o los movimientos bruscos para redirigir la dirección,

la hora del día…etc., calculan un índice de fatiga y sugieren al conductor una parada de descanso. Otros campos que se están desarrollando son la aplicación del *IoT* y del *BigData* en los vehículos, el almacenamiento de datos a gran escala permite a las empresas tomar decisiones y llevar a cabo proyectos exigentes y novedosos, fortaleciendo la conexión vehículo-conductor, como por ejemplo, la implementación de sistemas que recopilan información acerca de la música que el conductor frecuenta para posteriormente ofrecérsela. Este campo ha de recalcarse ya que el trabajo aquí presente abarcará en gran medida este concepto.

Se tratará de realizar un acercamiento al campo de los *Vehículos Conectados* o la conexión entre conductores y vehículos. En un futuro cercano se podrá controlar el nivel de combustible, la ubicación exacta del vehículo aparcado, gasolineras cercanas, la presión de los neumáticos…etc., mediante nuestro Smartphone o Tablet. El objetivo será conseguir una relación sólida con el vehículo, pudiéndose programar, por ejemplo, la climatización en el interior previamente a entrar en este, la visualización del estado de las diferentes piezas del vehículo e infinidad de aplicaciones que implican al conductor, vehículo y Smartphone.

1.3 Objetivos

El objetivo principal y general constituye la creación de un prototipo de vehículo inteligente a pequeña escala que sea capaz de ser controlado remotamente por un ser humano de manera sencilla.

Este proyecto, desde el comienzo tiene el principal objetivo personal de aprender, complementar conocimientos. También nos ha permitido conocer mucho más a fondo los siguientes campos:

- El S.O. **Android**, con sus respectivas herramientas para llevarlo a cabo como *Android Studio*, donde se realizará la aplicación para controlar el vehículo de manera remota, sencilla y planificada.

- La plataforma de desarrollo **Arduino**, para construir un dispositivo interactivo que pueda interactuar con el mundo real a través de entradas y salidas.
- El **IoT** y la tecnología de los **vehículos conectados**.
- Las tecnologías de comunicación, entre ellas el sistema ***Bluetooth***, que permitirá realizar la comunicación entre la placa Arduino y el Smartphone.

En cuanto al vehículo y la aplicación los objetivos que deberán cumplir serán los siguientes:

- Disponer de una interfaz clara y sencilla en la aplicación, de tal manera que esté todo ordenado y resumido, permitiendo y asegurando un uso correcto para cualquier tipo de usuario, ya sea experimentado o no.
- Ofrecer actividades que faciliten las tareas del ser humano y que estas funcionen correctamente. Se ha dado prioridad al funcionamiento adecuado de las actividades, postergando la adición de muchas actividades y que alguna pueda fallar en algún momento.
- Conseguir una aplicación sólida que permita al usuario utilizarla cuando desee y sin necesidad de ser administrada por ninguna entidad.
- Conseguir un código de programación claro y depurado, para que cualquier interesado pueda acceder fácilmente a él y añadir mejoras o nuevas funcionalidades.

1.4 Problemas iniciales

A lo largo del trabajo han ido surgiendo diversos problemas, que serán explicados a lo largo de este.

En primer lugar la elección de la versión de Android en la que posteriormente se trabajará. Más adelante la versión podrá modificarse en caso de que sea necesario.

Consultando la Figura 1.6 proporcionada por los desarrolladores de Android se puede observar cómo se distribuyen actualmente de manera aproximada las versiones de Android en los diferentes dispositivos de los usuarios.

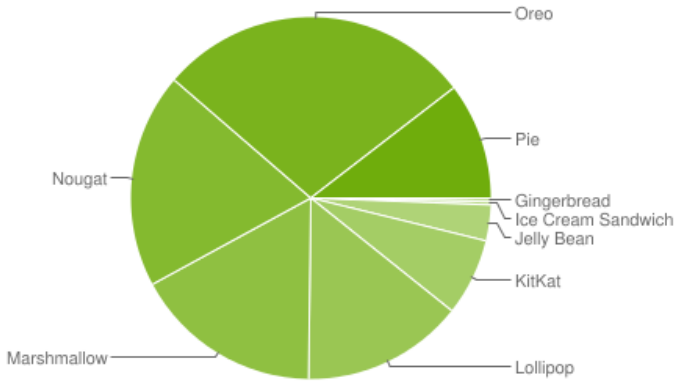


Figura 1.6 Distribución del uso de versiones de Android a 26/10/2018 [8]

Para la aplicación que se precisa será necesario un buen reconocimiento por voz y una buena capacidad de conexión y fluidez. Si se analiza la Figura 1.6, para este trabajo se escogerá una versión que sea lo más equilibrada posible para los usuarios. La versión que llevará a cabo esta aplicación será la **Api 21, versión 5.0 Lollipop** ya que los usuarios que dispongan de versiones posteriores (mayoría) podrán hacer uso sin problemas de la aplicación. De esta manera se abarca el mayor número de usuarios, siendo un 89,3% del total los que podrán hacer uso de la app. A continuación, en la Figura 1.7 se muestra la versión escogida en Android Studio.

```
defaultConfig {
    applicationId "com.example.asem.vi806_guillermo_fraga"
    minSdkVersion 21
    targetSdkVersion 28
    versionCode 1
    versionName "1.0"
    testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
}
```

Figura 1.7 Versión mínima requerida para la aplicación

Esta información puede obtenerla cualquier usuario desde su programa a través de *Gradle Scripts> build.gradle (Module: app).*

1.5 Herramientas utilizadas

Durante el desarrollo de este TFG se han utilizado las siguientes herramientas o elementos:

- Electrónica y relacionados:
    a) Placa ELEGOO Mega 2560.
    b) Smartphone Huawei P20 Lite.
    c) Cable USB 2.0 de tipo A a tipo B.
    d) Portátil HP Pavilion G6 Notebook. Características:
        I. Windows 8.1.
        II. Procesador Intel Core i7-3632QM.
        III. 8,0 GB RAM (7,89 utilizable).
    e) Cable USB tipo-C.
    f) Teclado y ratón.
    g) Protoboard.
    h) Cables de 0,5 mm de diametro.
    i) Módulo *Bluetooth* HC-05.
    j) Multímetro Digital HANMER S1.
    k) Resistencias.
    l) Diodo LED.
    m) Chasis de 2 capas y 4 ruedas.
    n) Motor Driver Shield con 2 L293D.
    o) 4 motores de corriente continua (3-6V).
    p) Sensor de temperatura y Humedad DHT11.

q) Escaner *EPSON V30* (Ilustraciones)

- Programación y software
  a) Android Studio 3.2.
  b) Arduino 1.8.9.
  c) Google Drive para almacenar el trabajo realizado.
  d) Paint, para editar imágenes y recortes.
  e) 7-zip para comprimir los archivos y manejarlos facilmente.
  f) Adobe Ilustrator CC 2018.

1.6 Estructura del proyecto

El desarrollo del Trabajo de Fin de Grado tendrá la siguiente estructura:

Capítulo 1: Introducción.

En este apartado se hace un breve analisis acerca de los conceptos que preceden y afectan al desarrollo del proyecto, otro análisis histórico de los vehículos inteligentes, los objetivos del proyecto, problemas iniciales y herramientas utilizadas.

Capítulo 2: Estado del arte

En este se analiza el estado y las tecnologías que preceden al presente Trabajo de Fin de Grado. En él se realiza un estudio acerca de los sistemas operativos para moviles que existen y han existido, de las plataformas para la creación de aplicaciones, de los métodos de comunicación y de las placas con microcontrolador que existen en el mercado.

Capítulo 3: Métodos

Este capítulo engloba el desarrollo de la aplicación y del prototipo del vehículo, el cual se divide en 3 subcapítulos:

- **Vehículo inteligente**:

  En él se analizan y desarrollan los procesos aplicados en la electrónica del proyecto, así como su relación con Arduino.

- **Aplicación móvil**:

  En este subcapítulo se muestra el desarrollo de la aplicación hasta el momento en el que se realiza un intercambio de datos entre el Smartphone y Arduino, es decir, desde la portada, hasta el menú principal, pasando por la obtención de la dirección MAC de nuestro módulo Bluetooth.

- **Intercambio de datos**:

  Aquí se analiza el desarrollo de las tres funciones principales de nuestra aplicación. Es por esto por lo que este subcapítulo se divide en 3 apartados; el primero, **Estado del vehículo**, que engloba la obtención de datos desde el prototipo para mostrarlos el Smartphone; el segundo, **Control del vehículo**, donde se desarrolla el control remoto del vehículo y por último la **Localización**, donde se obtiene la localización del vehículo a través del Smartphone.

Capítulo 4: Conclusión

En este capítulo se desarrolla la conclusion final del proyecto realizado, analizando los objetivos cumplidos y positivos junto con los negativos.

CAPITULO 2. ESTADO DEL ARTE

A continuación, en el estado del arte se analizan las investigaciones que preceden a los campos que han sido necesarios para desarrollar el proyecto.

En primer lugar se hace un estudio acerca de los diferentes **sistemas operativos** para móviles que han quedado en desuso con el transcurso del tiempo y los que han conseguido marcar tendencia en el mercado y se encuentran actualmente operativos. También se realiza un estudio detallado acerca del sistema operativo Android, ya que es el sistema operativo sobre el que se realiza la aplicación.

En segundo lugar se profundiza acerca de las **plataformas** que se disponen actualmente para crear aplicaciones móviles y de la evolución de éstas.

En tercer lugar se analiza los diferentes **métodos de comunicación** que han existido y la elección del *Bluetooth* como base del proyecto.

En cuarto y último lugar se hace un análisis sobre las diferentes alternativas acerca de las **placas** con microcontrolador y código abierto y la elección de Arduino.

2.1 Sistemas operativos y su progreso en el mercado

Un sistema operativo es un programa o conjunto de programas que realiza diferentes tareas para gestionar todos los recursos del sistema informático tanto de hardware (pantalla, disco duro, periféricos…) como de software (instrucciones y programas). Sirve de vínculo entre el usuario y el dispositivo [9].

Existen sistemas operativos para cualquier dispositivo con el que se pueda interactuar, los más conocidos y utilizados son los destinados a las computadoras, a los dispositivos móviles y a los relojes inteligentes.

En la siguiente figura se muestra mediante un gráfico la distribución de la cuota de mercado de los sistemas operativos móviles desde 2009 a 2019 en todo el mundo:

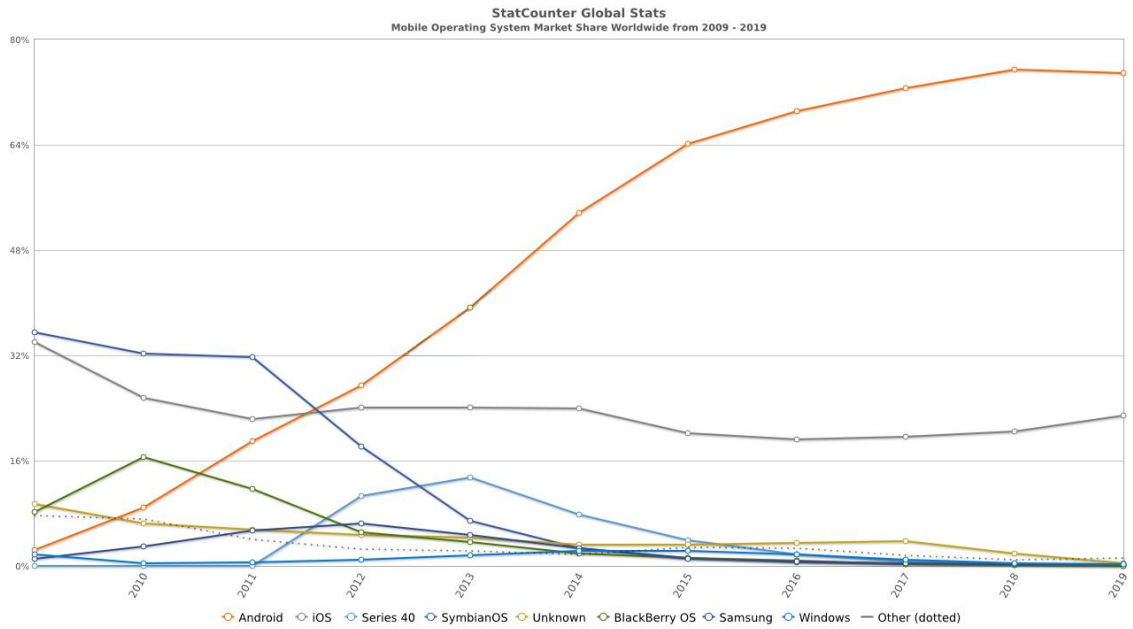


Figura 2.1 Evolución de cuota de mercado en SO móviles [10]

Se observa como a principios el mercado estaba distribuido de manera completamente diferente a la actualidad. En 2009 Android comenzaba su expansión en el mercado, pero había otros sistemas operativos que competían como *iOS*, *Samsung*, *Blackberry*, *Windows Mobile* o *Symbian*. A continuación se hace una breve introducción de algunos de los sistemas operativos móviles que han cobrado importancia en la historia.

2.1.1 Samsung

La mayoría de dispositivos Samsung disponen del sistema operativo Android, pero a parte de este, llevan a cabo un sistema operativo propio llamado *Tizen*, basado en Linux [12], el cual se encuentra incorporado en diferentes dispositivos móviles y *Smartwatches.*

Figura 2.2 Logotipo de Tizen [12]

2.1.2 Symbian

Este sistema operativo comenzó en 1997 con el nombre de EPOC, era propiedad de la empresa Nokia y colaboró con otras compañías como Sony, Samsung, Motorola, Panasonic…etc. Lideró el mercado hasta el año 2011, donde ya comenzó su decadencia, dominando entonces Android e iOS y Nokia fabricó el último modelo con éste sistema operativo [11].

Figura 2.3 Logotipo de Symbian [11]

2.1.3 Windows Mobile

Windows Mobile se lanzó con su primer dispositivo en el año 2000 y está basado en el sistema operativo y *API*[1] desarrolladas por Microsoft en su sistema operativo llamado Microsoft CE. Fue diseñado para teléfonos inteligentes y dispositivos móviles, posteriormente pasó a denominarse Windows Phone [x].

En la figura 2.4 se muestra en primer lugar el logotipo de Windows Mobile y en segundo lugar el de Windows Phone.

Figura 2.4 Logotipo de Windows Mobile y Windows Phone [13]

[1] Conjunto de código que se puede emplear para que varias aplicaciones se comuniquen entre ellas. Véase https://neoattack.com/neowiki/api/

2.1.4 Blackberry

Esta compañía se fundó en 1999 y anteriormente se la conocía como RIM (*Research In Motion*), lanzó entonces su primer modelo, donde se podía gestionar el correo electrónico, agenda, enviar y recibir páginas de internet y enviar mensajes de texto. Su sistema operativo permitía realizar multitareas y manejar el hardware que incorporaban sus productos como el teclado *QWERTY* o la *trackball* (Bola de seguimiento) [14].

Figura 2.5 Logotipo de Blackberry [14]

En cuanto al mercado que domina actualmente en los sistemas operativos móviles se encuentran iOS y Android. En la figura 2.6 se encuentra un gráfico estadístico donde se reflejan las ventas mundiales de estas dos compañías desde el 2009 al 2018:

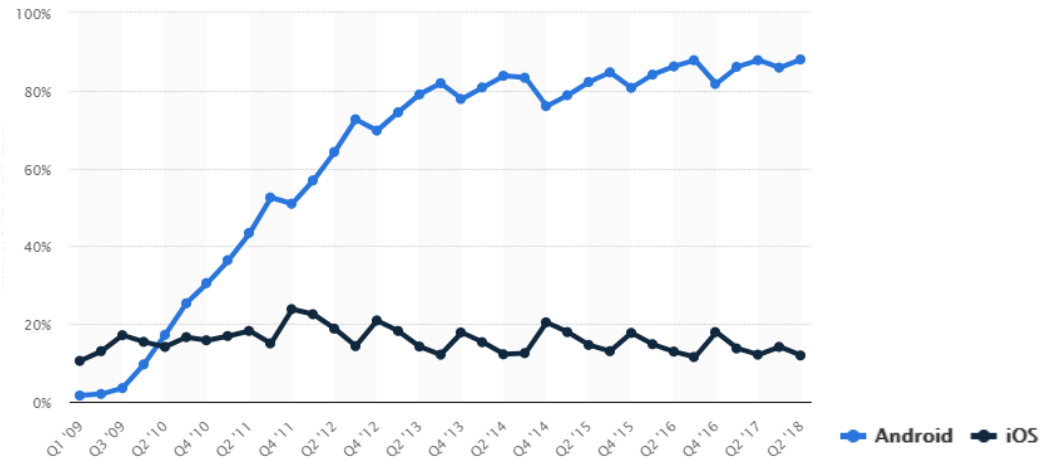


Figura 2.6 Ventas de dispositivos iOS y Android [15]

Se puede observar como el mercado se reparte entre estas dos empresas; cuando una de ellas introduce un nuevo modelo las ventas se disparan y cae la contraria y así de

manera inversa. A continuación se hace un análisis más detallado sobre estas dos compañías:

2.1.5 iOS

iOS es el sistema operativo que utiliza la empresa **Apple Inc**., se dió a conocer en el año 2007 en la convención anual conocida como *Macworld Conference & Expo* donde esta mostraba sus nuevos productos. Este sistema operativo deriva de lo que en un principio se conocía como mac OS X, desarrollado desde 2001 y utilizado en las computadoras de esta empresa. En esta convención, el fundador de la empresa *Steve Jobs* presentó el primer *iPhone*, el primer teléfono inteligente que disponía de pantalla táctil, gestos *multi-touch* y una interfaz muy visual [16].

La introducción de este *Smartphone* al mercado supuso una revolución en el concepto de teléfono inteligente que se tenía entonces, acabando con el sistema de teclado *T9* y *QWERTY*. Desde entonces la empresa ha ido actualizando continuamente su software, añadiendo y mejorando funciones para seguir dominando en el mercado.

Figura 2.7 Logotipo de Apple Inc [16]

2.1.6 Android

Android. Inc es creada en 2003 y comprada por Google en julio de 2005.

En 2007 ante la innovadora aparición del *iPhone* se crea un consorcio llamado *Open Handset Alliance* donde se alían grandes empresas como *Samsung*,

Motorola, *HTC*, *Google*…etc. para desarrollar un sistema operativo de código abierto. Finalmente se acordó en que sería Android el sistema operativo que se desarrollaría. En 2008 se lanza el primer *Smartphone* con este sistema operativo, con la versión 1.0 *Apple Pie*, pudiendo interactuar con los productos más populares de Google [17].

Desde entonces, Android ha estado trabajando continuamente hasta conseguir ser el sistema operativo más utilizado del mundo, repartiéndose el mercado con iOS, siendo de Android aproximadamente el 80% del mercado actual.

Figura 2.8 Logotipo de Android [17]

2.1.6.1 Arquitectura

Este software posee una **arquitectura** completa, se trata de un software de código abierto [18] administrado por capas, donde cada capa hace uso de los servicios proporcionados por la capa anterior. A continuación se hace un análisis de dichas capas desde la base (Kernel de Linux) al servicio final (Aplicación). Ver figura 2.9.

- **Kernel de Linux**:

  El Kernel de Linux o raíz de Linux es la base de la plataforma, permite a Android aprovechar las funciones de seguridad claves y permite contener los *drivers* o controladores necesarios para que cualquier componente hardware funcione correctamente.

- **Android Runtime** (Tiempo de ejecución de Android) y **librerías**:

  Este es el entorno de ejecución, que abarca el tiempo de compilación y ejecución de las aplicaciones y las librerías utilizadas por Android. Previamente a la versión 5.0 *Lollipop* se utilizaba una máquina virtual *Dalvik*, que trabajaba con un sistema *JIT* (Just-in-time o Justo a tiempo), es decir, que compilaba al tiempo que ejecutaba, pero a partir de entonces se utiliza otra máquina virtual llamada *ART* (Android Runtime). Esta trabaja con un sistema *AOT* (Ahead-of-time o compilación anticipada), que permite compilar en tiempo de instalación y no de ejecución, obteniendo más rendimiento pero requiriendo más espacio, entre otros beneficios.

- **Framework de las aplicaciones**:

  En el Framework o Armazón de las Aplicaciones se encuentran las *API*, es decir, el conjunto de herramientas y funciones necesarias para el desarrollo de una aplicación. Ya sea una aplicación predeterminada del dispositivo, o desarrollada por Google o por un tercero, todos utilizan las mismas *API*.

- **Aplicaciones**:

  Es el nivel que cualquier propietario de un *Smartphone* conoce y el más alto de todos. Una aplicación engloba todos los niveles nombrados anteriormente, es decir, posee las librerías, las *API*, los servicios…etc. En Android se incluye un conjunto de apps centrales para correo electrónico, mensajería SMS, calendarios y contactos entre otros.

A continuación se muestra la figura 2.9 donde se observa de manera gráfica los distintos niveles que componen la arquitectura de este sistema operativo:

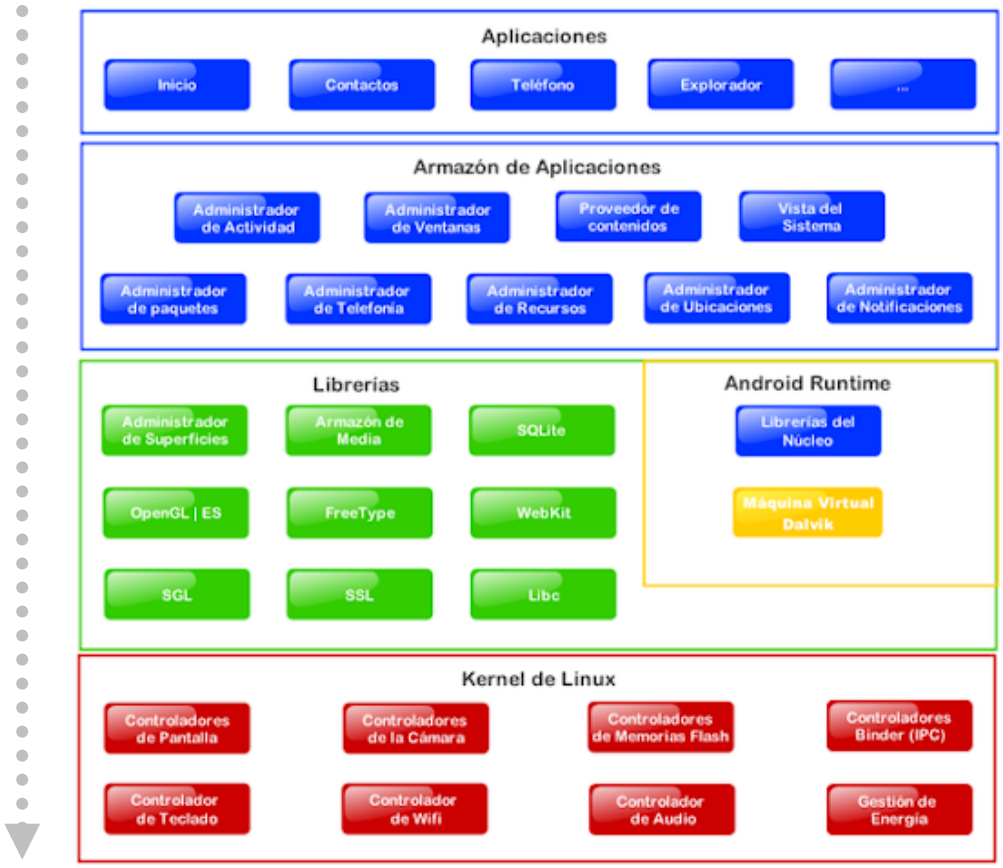


Figura 2.9 Arquitectura del SO Android [19]

2.1.6.2 Versiones Android

Desde 2008 hasta la actualidad Android ha ido actualizándose para mantenerse en el primer puesto del mercado. Sus actualizaciones, van desde la 1.0 *Apple Pie* hasta la 10.0 *Q*, todas ellas se definen mediante el nombre de un postre o dulce. Cada versión actualizada incluye nuevas funciones que no tienen las anteriores, por lo que una aplicación creada con una versión determinada podrá ser ejecutada por un *Smartphone* con una versión posterior, pero quizás no por uno con versión más antigua, dependiendo de si tiene o no los requisitos que utiliza.

En la siguiente tabla se observa el historial de todas las versiones Android y su traducción.

Letra	Nombre	Versión	Traducción
A	Apple Pie	1.0	Tarta de manzana
B	Banana Bread	1.1	Pan de plátano
C	Cupcake	1.5	Magdalena
D	Donut	1.6	Donut
E	Éclair	2.0 / 2.1	Palo de crema
F	Froyo	2.2	Yogur helado
G	Gingerbread	2.3	Pan de jengibre
H	Honeycomb	3.0 / 3.1 / 3.2	Panal
I	Ice Cream Sandwich	4.0	Sándwich de helado
J	Jelly Bean	4.1 / 4.2 / 4.3	Gominola
K	KitKat	4.4	Kit Kat
L	Lollipop	5.0 / 5.0.1 / 5.0.2 / 5.1	Piruleta[55]
M	Marshmallow	6.0 / 6.0.1	Malvavisco[56]
N	Nougat	7.0 / 7.1 / 7.1.1 / 7.1.2	Turrón
O	Oreo	8.0 / 8.1	Oreo
P	Pie	9.0	Pastel
Q	Q	10.0	Q

Tabla 2.1 Historial de versiones Android [20]

2.2 Plataformas para el desarrollo de aplicaciones

Con el amplio futuro y el auge de las aplicaciones Android han surgido múltiples entornos de programación para estas. A continuación se hace un análisis sobre los entornos de programación de aplicaciones más utilizados en la actualidad:

- **Eclipse**:

  Eclipse es una plataforma de software con diferentes herramientas de programación de código abierto que fue creada por la empresa multinacional *IBM* en 2003 [21]. En un principio esta plataforma fue el IDE (Entorno de desarrollo integrado) recomendado, *Google* hizo de esta plataforma su IDE oficial para el desarrollo de aplicaciones, pero este puesto fue sustituido por Android Studio en 2013.

  Es un entorno de programación que permite desarrollar en varias plataformas, ya que no es un entorno destinado exclusivamente a aplicaciones móviles. Para poder utilizarlo con este fin se debe instalar un

plug-in[2] llamado *ADT* previamente, que extiende las capacidades de eclipse para poder trabajar con las funcionalidades que se requieren. Este entorno trabaja con el lenguaje de programación Java, aunque puede extenderse y utilizar otros lenguajes como C/C++ o Phyton.

Este entorno al soportar múltiples plataformas se convierte, de cara al desarrollo de aplicaciones, en un entorno más lento. Las otras características que hacen de Android Studio el IDE seleccionado para este Trabajo de Fin de grado se analizan más adelante en el análisis de dicho entorno.

Figura 2.10 Logotipo de eclipse [21]

- **Xamarin**:

Xamarin, fundada en 2011, es una compañía de software estadounidense que fue adquirida por Microsoft en 2016. La diferencia de este entorno con el resto es que se trata de un entorno que permite trabajar con aplicaciones para plataformas iOS, Android y Windows Mobile en el mismo entorno. Utiliza el lenguaje de programación C#, que se basa en los lenguajes C y C++, pero puede usar otros con la adición de bibliotecas [22].

Figura 2.11 Logotipo de Xamarin [23]

[2] Aplicación que se relaciona con otra y añade una funcionalidad nueva específica. Véase https://definicion.de/plugin/

- **Appcelerator Titanium**

  Appcelerator Titanium [24] es un entorno de programación de código abierto creado en 2008 que a principios estaba destinado al desarrollo de aplicaciones de escritorio para múltiples plataformas, pero un año más tarde se agregó soporte para el desarrollo de aplicaciones móviles para iOS y Android y posteriormente se añadió soporte también para Windows Mobile. Este entorno de programación usa exclusivamente Javascript[3], que mejora su rendimiento, pero también tiene inconvenientes, como no disponer de un HTML inicial donde añadir controles y componentes visuales, si no que estos han de añadirse manualmente con Javascript; o que se requiere de una computadora Mac y el entorno *Xcode* para empaquetar aplicaciones iOS.

Figura 2.12 Logotipo de Appcelerator [25]

- **Android Studio**

  Android Studio fue presentado en 2013 en la conferencia anual Google I/O como el IDE oficial para el desarrollo de aplicaciones Android. Está basado en el software *Intellij IDEA*, de la compañía Jetbrains y se encuentra disponible para las plataformas Microsoft Windows, Mac y Linux de forma gratuita [27].

  La primera versión aún era un poco inestable, pero ha ido actualizándose y mejorando continuamente hasta disponer, actualmente, de 37 versiones desde que la versión 1.0 se dió a conocer. Esto hace de Android Studio un entorno

[3] No confundir con Java. Sirve para ejecutar *scripts* (archivo de órdenes), se integra directamente en las páginas HTML y es interpretado por el navegador. Véase https://www.gestiopolis.com/diferencias-java-javascript/

actualizado frente a los cambios en los dispositivos móviles. A continuación se muestran algunas de las características que hacen de este entorno el IDE oficial y el seleccionado para el Trabajo de Fin de Grado:

- Utiliza el sistema de construcción *Gradle*, que construye sobre los conceptos de *Apache ant y Apache Maven* que son los utilizados en entornos como Eclipse. De esta manera se convierte en el sistema de automatización de compilación más estable, ya que recoge las ventajas de los otros sistemas [26]. En la figura 2.13 se muestran las características que recoge este sistema:

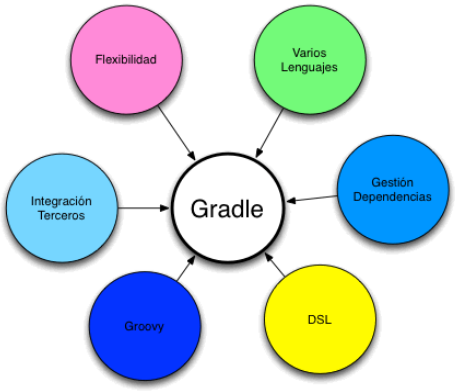


Figura 2.13 Caracteristicas del sistema Gradle [26]

- Editor de diseño enriquecido que permite a los usuarios disponer de una interfaz muy visual y fácil de utilizar.
- Dispone de una consola muy útil para el desarrollador, ya que muestra diferentes consejos de optimización, ayuda para la traducción, corrección de errores, estadísticas de uso…etc.
- Soporte para programar aplicaciones para *Android Wear* (dispositivos corporales).

A continuación se muestra el logotipo de Android Studio:

Figura 2.14 Logotipo de Android Studio [27]

2.3 Métodos de comunicación

En este apartado se hace una introducción breve acerca de los tipos de comunicación, las topologías de red que existen y la comunicación escogida.

2.3.1 Tipos de comunicación

Existen tres métodos de transmisión para enviar y recibir mensajes: Unicast, Broadcast y Multicast [28].

- Unicast:

  Este método de transmisión *one-to-one* (uno a uno) se utiliza para un único emisor y un único receptor. El cliente lanza una petición, el servidor la analiza, procesa y envía una respuesta a éste.

- Broadcast:

  Utiliza el método *one-to-all* (de uno a todos), es decir, envía la información a todos los nodos de la red, independientemente de si uno de los clientes no necesita o requiere de dicha información.

- Multicast:

  Multicast utiliza el método *one-to-many* (de uno a muchos), es decir, se envía la información a los nodos que la requieran. También puede darse aquí el método *many-to-many* (de muchos a muchos)

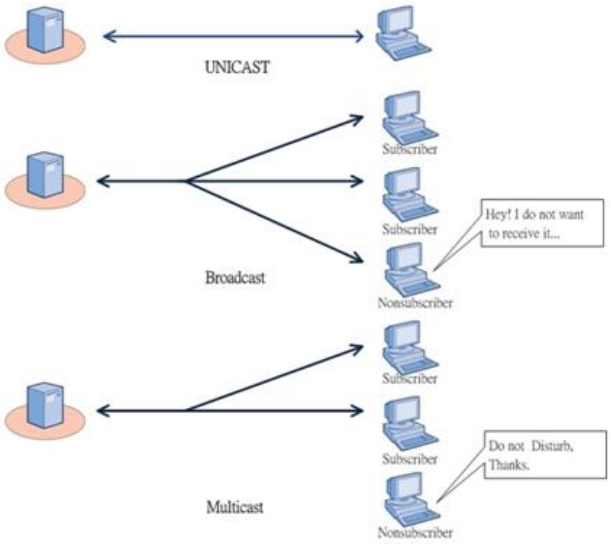


Figura 2.15 Tipos de comunicación [29]

En la figura anterior se muestran de manera visual los diferentes tipos de comunicación.

2.3.2 Topologías de red

Constan de los mapas lógicos de una red y se verán únicamente los más comunes sin entrar en detalles de las ventajas y desventajas que suponen cada uno. La topología **punto a punto** consta de los dos puntos finales de una comunicación, es la más simple de todas; la topología **bus**, se usa principalmente en redes LAN (*Local Area Network* o red de área local), donde existe un cable principal por el que circula la información en ambos sentidos hasta el destinatario deseado; la topología **estrella**, que dispone de un nodo central al que se conectan los demás nodos y la topología **anillo**, que supone una topología bus en circuito cerrado donde la información viaja en una dirección hasta dar con el destinatario.

En la figura 2.16 se encuentran las diferentes tipologías que existen:

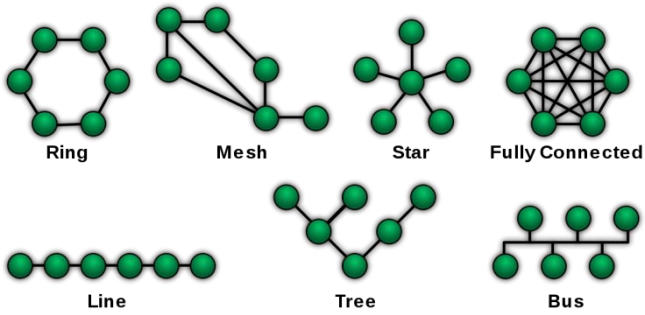


Figura 2.16 Topologías de red [30]

2.3.3 Redes inalámbricas

Tras la anterior introducción a las redes y comunicación, se concluye que, para este Trabajo de Fin de Grado, necesitamos trabajar con la topología punto a punto, del *Smartphone* a nuestro vehículo personal. También necesitamos una comunicación inalámbrica, ya que el objetivo es trabajar a distancia con el vehículo y reducir los tiempos de espera del usuario; para esto, las opciones más viables y utilizadas son la tecnología Wi-Fi y la tecnología Bluetooth. A continuación se hace un análisis comparativo de ambas tecnologías.

2.3.3.1 Tecnología Bluetooth

La tecnología *Bluetooth* [31] es un modo de comunicación que se descubrió en 1994 por parte de la empresa *Ericcson*, es una tecnología muy barata y económica, pero a la vez, de corto alcance. En ella se pueden diferenciar cuatro clases, siendo la Clase 1 la de mayor rango (100 metros y consumo de 100 mW) y la Clase 3 la de menor. En la siguiente tabla se hace un resumen de estas clases:

Clase	Potencia Máx. (mW)	Potencia Máx. (dBm)	Rango
Clase 1	100 mW	20 dBm	100 metros
Clase 2	2.5 mW	4 dBm	20 metros
Clase 3	1 mW	0 dBm	1 metro

Tabla 2.2 Clases de Bluetooth

En su desarrollo se han forjado 5 versiones:

- Bluetooth 1.0:

La primera versión de todas, dio lugar a muchas complicaciones y actualmente se encuentra en desuso. Le siguieron las versiones 1.1 y 1.2, donde se corrigieron cantidad de errores y se mejoró considerablemente, entre otras, la velocidad de transmisión, llegando a ser de 721 *kbps*. También se añadió el concepto *Discovery*, es decir, que pudiese detectar otros dispositivos Bluetooth.

- Bluetooth 2.0:

Se lanza en 2004 e introduce el sistema EDR (*Enhanced data rate*, tasa de datos mejorada) para incrementar la rapidez, pasando a ser la tasa de transferencia de datos de 2.1 *Mbps*. Posteriormente surge la versión 2.1 en 2007, que mejora el emparejamiento entre dispositivos y la seguridad de la comunicación.

- Bluetooth 3.0:

Se introduce el sistema *HS* (*High Speed* o Alta velocidad), que permite utilizar una velocidad de transmisión de 24 *Mbps*, permitiendo el envío y recepción de archivos con mucha cantidad de datos como archivos de vídeo o imagen.

- Bluetooth 4.0:

Se lanza en 2010, incorpora los sistemas de *Bluetooth* clásico, ya adoptado en las versiones anteriores, *Bluetooth HS* y además se introduce un novedoso sistema llamado *BLE* (Bluetooth Low Energy) que permite realizar enlaces sencillos en

circuitos de muy baja potencia. Posteriormente se actualiza a la 4.1 y 4.2. Es la versión utilizada por los *Smartphones* actuales y la velocidad alcanza los 32 *Mbps.*

- Bluetooth 5.0:

Se encuentra disponible desde finales de 2016, tiene el doble de velocidad, más fiabilidad y mayor rango de cobertura. Sus cualidades son próximas a las de la tecnología Wi-Fi y se incorpora un novedoso sistema llamado "balizamiento offline", que permitirá localizar nuestro dispositivo al estar cerca de diferentes comercios para mejorar la efectividad de la publicidad, entre otros servicios.

Nuestro módulo Bluetooth *HC-05* dispone de la versión 2.0, que será suficiente para llevar a cabo el proyecto y suplir las necesidades de éste.

En esta tecnología existen dos tipos de dispositivos, el **maestro** y el **esclavo**. El maestro es quien se encarga de iniciar la comunicación y gestionar las frecuencias y los esclavos son quienes transmiten información de acuerdo con los ciclos de los maestros [32].

La **topología** de red que utiliza la tecnología Bluetooth se denomina *piconet*. Esta topología utiliza a su vez el sistema punto a punto comentado anteriormente, siendo, el máximo número de dispositivos conectados de 8 por piconet y hasta 10 piconets. La red que se crea cuando un dispositivo se conecta a dos maestros (1 por cada piconet) se denomina *scatternet*. A continuación se muestra en la figura un gráfico para comprender mejor este concepto:

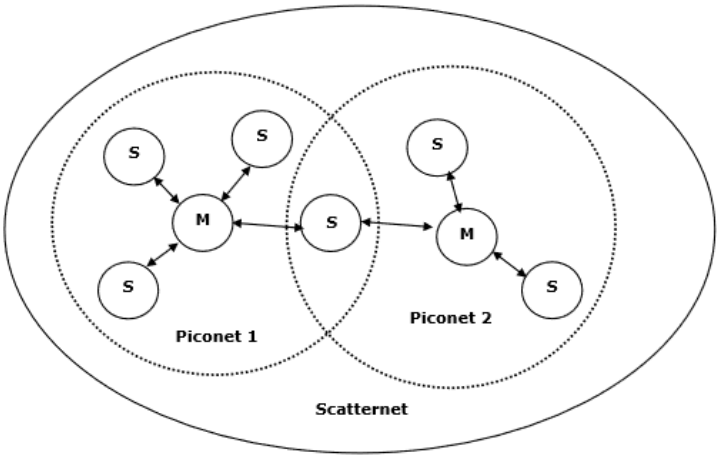


Figura 2.17 Redes piconet [33]

Tras analizar la topología de la red Bluetooth se estudia la forma de transmisión de datos de esta tecnología. Para comprenderlo mejor se adjunta la siguiente figura:

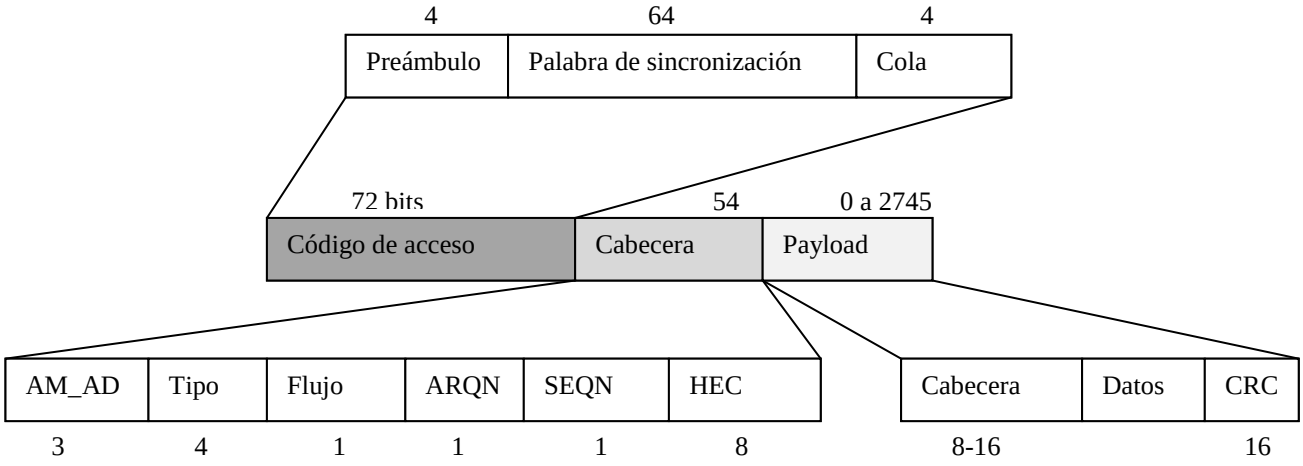


Figura 2.18 Transmisión de paquetes ACL en Bluetooth

La tecnología Bluetooth se comunica a través de paquetes. Esta comunicación puede realizarse a través de un enlace síncrono para la comunicación por voz o a través de un enlace asíncrono para el tráfico de paquetes de datos (ACL). El formato de cada paquete ACL, que es el que se utilizará en este proyecto, sigue el esquema de la figura anterior, diferenciándose en tres zonas principales. En primer lugar se encuentra el código de acceso, que deriva de la identidad

maestra, formado por 72 bits y encargado de la sincronización y de facilitar la identificación y compensación. Los paquetes que son enviados al mismo canal de la piconet tienen el mismo código de acceso. En segundo lugar se encuentra la cabecera, que contiene información del control de enlace y sus 54 bits se dividen en 6 *slots*, entre ellos el del Flujo que notificará cuando el *buffer* del receptor está lleno para dejar de transmitir. Por último el *Payload* o campos de datos, que contiene la información a transmitir [34].

2.3.3.2 Comparativa Wi-Fi - Bluetooth

La tecnología Wi-Fi o *Wireless Fidelity* fue creada a partir de la unión de varias empresas en una alianza llamada WECA (*Wireless Ethernet Compatibility Alliance*), donde se realizaron diversas investigaciones desde 1999 para fomentar la tecnología inalámbrica. El objetivo del Wi-Fi consiste en poder conectarse a internet o a otras redes, y para esto es necesario disponer de un dispositivo intermedio llamado router [35]. A continuación se muestra una tabla comparativa entre esta tecnología y el Bluetooth:

	Bluetooth	Wi-Fi
Frecuencia	2.4 GHz	2.4 / 3.6 / 5 GHz
Ancho de banda	1 - 24 Mbps	1 Gbps
Seguridad	Baja	Moderada
Rango	Hasta 30 metros (puede llegar a 100 según la clase)	Hasta 300 metros
Consumo	Bajo	Elevado
Facilidad de uso	Sencillo	Complejo

Requisitos de hardware	Adaptador Bluetooth	Adaptador inalámbrico y router
Numero de dispositivos	Reducido	Alto

Tabla 2.3 Comparativa de tecnología Wi-Fi – Bluetooth

Como se puede observar en la tabla anterior, las diferencias entre estas dos tecnologías son claras. Las más notables se encuentran en el ámbito de la seguridad, rango y consumo. La seguridad de la tecnología Bluetooth es muy baja, si se requiere de un protocolo más seguro y fiable ha de utilizarse la tecnología Wi-Fi que dispondrá a la vez de mayor rango. Pero no por estas características una tecnología es mejor o peor que otra, sino que, cada una de ellas está destinada a actividades diferentes; si se opta por una conexión más rápida probablemente será una conexión menos segura. El Bluetooth está destinado para dispositivos que usualmente se encuentran cerca del usuario, y es por ello por lo que el rango de esta tecnología es bajo y para dispositivos que no requieran de una seguridad alta. En cuanto al consumo se observa como la tecnología Wi-Fi requiere de uno superior al del Bluetooth. La tecnología Bluetooth en un principio si consumía mucha energía, pero con la mejora y optimización de ésta se ha conseguido una tecnología de bajo consumo, permitiendo así una alta autonomía de los dispositivos que gozan de ésta. Otra de las características que diferencian notablemente a estas dos tecnologías es el número de dispositivos que pueden enlazarse, ya que el Wi-Fi permite que varios dispositivos se conecten a una red simultáneamente mientras que el Bluetooth únicamente permite la conexión entre pocos dispositivos.

- **Conclusión**

Por estas comparativas, se ha decidido llevar a cabo el proyecto utilizando la tecnología Bluetooth, ya que, en este caso, al tratarse de un prototipo a pequeña escala, se ha de utilizar una tecnología de bajo consumo y fácil de utilizar. Como en este caso no se requiere conexión a internet y únicamente queremos conectar

el *Smartphone* con el vehículo no será necesaria otro tipo de tecnología. Por último, al tratarse de un prototipo, el rango que se requiere tampoco es muy alto, con el Bluetooth será suficiente.

2.4 Placas de hardware y software abierto (SBC)

El concepto del "diseño abierto" se remonta a los años 1998-1999, cuando se establece la ODF (*Open Design Fundation*) como una corporación sin fines de lucro para desarrollar esta definición. Este concepto fue expandiéndose y surgió entonces la idea de *Open Design Circuits*, destinado a la creación de hardware y software libre [36]. Se analizan algunas de las ventajas y desventajas de utilizar código abierto en un proyecto:

- **Ventajas**:

  Bajo coste, ya que no existen costes de licencias ni mantenimientos, utilizan tecnologías actualizadas, hay más facilidad de personalización y existe detrás una comunidad que permite resolver dudas, mejorar el producto y evitar errores.

- **Desventajas**:

  No hay garantías ni responsables ante un mal funcionamiento, es ilegal o costosa la adaptación de este a una necesidad particular.

Son mayores los beneficios adquiridos frente las desventajas, por ello, con este nuevo concepto, surgieron muchas empresas destinadas a crear hardware y software libre.

Muchas de ellas están enfocadas a la creación de placas SBC (*Single Board Computer*), es decir, ordenadores de una única placa reducida. A continuación se verán las dos empresas destinadas a la creación de éstas placas más conocidas:

2.4.1 Arduino

Esta compañía nació en Italia alrededor de los años 2003-2005 y fue la primera empresa que promovió este tipo de placas de bajo costo y fáciles de utilizar. El objetivo era comercializar PCB's (*Printed Circuit Board*) con un microcontrolador, un entorno de desarrollo integrado y una biblioteca de funciones. Estas placas estarían principalmente destinadas para estudiantes o para personas sin muchos conocimientos técnicos.

Como se ha comentado anteriormente, es fabricante de software y hardware libre, por lo que en su página web se puede disponer de toda la información acerca de estos, estando al alcance de cualquier usuario o desarrollador [37].

A continuación se muestra una tabla con los diferentes modelos existentes de esta compañía y sus características:

Modelo	Microcontrolador	Voltaje de entrada	Voltaje del sistema	Frecuencia de Reloj	Digital I/O	Entradas Analógicas	PWM	UART	Memoria Flash
Arduino Due	AT91SAM3X8E	5-12V	3.3V	84MHz	54	12	12	4	512Kb
Arduino Leonardo	ATmega32U4	7-12V	5V	16MHz	20	12	7	1	32Kb
Arduino Uno - R3	ATmega328	7-12V	5V	16MHz	14	6	6	1	32Kb
Arduino Pro 3.3V/8MHz	ATmega328	3.35 -12V	3.3V	8MHz	14	6	6	1	32Kb
Arduino Pro 5V/16MHz	ATmega328	5 - 12V	5V	16MHz	14	6	6	1	32Kb
Ethernet	ATmega328	7-12V	5V	16MHz	14	6	6	1	32Kb
Arduino Mega 2560 R3	ATmega2560	7-12V	5V	16MHz	54	16	14	4	256Kb
Arduino Mini 05	ATmega328	7-9V	5V	16MHz	14	6	8	1	32Kb
Arduino Pro Mini 3.3V/8MHz	ATmega328	3.35 -12V	3.3V	8MHz	14	6	6	1	32Kb
Arduino Pro Mini 5V/16MHz	ATmega328	5 - 12V	5V	16MHz	14	6	6	1	32Kb
Arduino Fio	ATmega328P	3.35 -12V	3.3V	8MHz	14	8	6	1	32Kb

Tabla 2.4 Modelos y características de Arduino [38]

Los precios de los modelos varían desde los 8 a los 67 € en la página oficial de Arduino.

2.4.2 Raspberry Pi

Fue fundada en 2009, pero no se lanzaron a la venta a nivel mundial hasta 2012. Raspberry Pi es un ordenador de placa reducida con un único microprocesador, desarrollado en Reino Unido que ha ido evolucionando y mejorando sucesivamente, hasta disponer actualmente de 9 modelos y versiones. La memoria RAM de estas placas varía entre 256 MB y 4 GB dependiendo del modelo; no disponen de disco duro ya que utilizan tarjeta SD y los modelos actuales disponen de dos puertos USB, uno de Ethernet y una salida HDMI [39]. Disponen de menos modelos que la compañía anterior y sus precios oscilan entre los 6 y los 68 € dependiendo del modelo. A continuación se muestra el logotipo de la fundación y el modelo *Raspberry Pi 2 modelo B*:

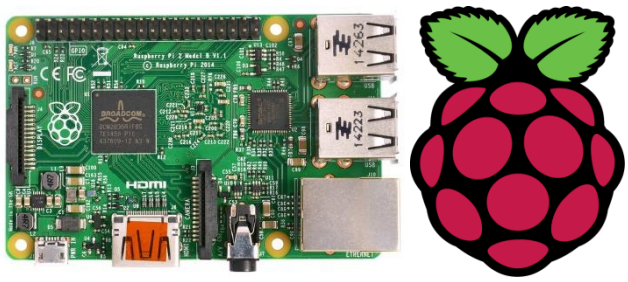


Figura 2.19 Raspberry Pi 2 modelo B y logotipo de la fundación [39]

Existen otras empresas fabricantes de este tipo de productos como son **Texas Instruments** con su placa *Beagleboard* y *Beaglebone*, **Microsoft** con su placa *Sharks Cove*, **Intel** con su *Minnowboard*, **Libelium** con su placa Waspmote…etc.

En la siguiente tabla se muestra una comparativa de dos modelos de características similares de Raspberry Pi y Arduino:

	Arduino Uno	Raspberry Pi 1 Modelo A/B
Funcionamiento	Microcontrolador ATMEGA 328	Microprocesador de 256/512 MB de RAM
Sistema operativo	Multiplataforma	Propio: *Raspbian*
Internet	Necesitaría una Shield externa para el acceso	Puerto de comunicación Ethernet
Salidas	USB, 19 pines A/D	USB, HDMI, RCA, Audio y 40 pines A/D
Velocidad	16 MHz	700 MHZ
Destino	Educación electrónica	Educación informática
Precio (€)	10	35/50

Tabla 2.5 Comparativa de placas Arduino y Raspberry

Tras esta comparativa y un análisis de las necesidades del Trabajo de Fin de Grado se ha decidido optar por una placa Arduino, debido a que el proyecto está enfocado a la práctica de electrónica y no directamente a la informática; al precio, ya que estas son más asequibles y existe mayor rango; a la disponibilidad, ya que en el mercado se encuentran más modelos de Arduino y a las necesidades, que se explicarán más adelante.

CAPÍTULO 3. MÉTODOS

Para llevar a cabo un orden lógico y ordenado en este apartado del documento, se analizará, por un lado, el campo relacionado con Arduino y los componentes electrónicos, es decir, el vehículo inteligente, y por otra parte el campo de Android y su programación, la aplicación móvil.

3.1 Vehículo inteligente

El prototipo del vehículo utilizado en este trabajo se sustentará por **Arduino**, como se ha comentado con anterioridad. Como ya se ha realizado una breve introducción de dicha compañía anteriormente, se continuará desde esa base.

En este caso se ha utilizado la placa de desarrollo Mega 2560. La opcion más económica consta de la placa Arduino UNO, pero esta cuenta unícamente con 14 pines digitales (6 salidas PWM), 6 entradas analógicas y 32k de memoria flash.

Para asegurar una cantidad suficiente de entradas y salidas disponibles y para introducir posibles funciones nuevas en un futuro se ha optado por un modelo más completo, la Mega 2560 de la marca ELEGOO, la cual se puede observar en la figura 3.1, basada en el microcontrolador ATMega2560.

Figura 3.1 Placa Mega2560 de la marca ELEGOO [40]

Tal y como se puede observar en la figura anterior, esta placa dispone de 54 pines de entrada o salida digitales de las cuales 15 pueden utilizarse como salidas PWM, 16 entradas analógicas 4 UART's para conectar módulos de comunicación, conexión USB, botón de reseteo, jack para alimentación DC, conector ICSP y el microcontrolador nombrado anteriormente.

Para llevar a cabo este proyecto se hará uso del programa Arduino versión 1.8.9. Se hará un repaso del funcionamiento y de los aspectos relevantes de este programa.

Arduino se nos presenta como en la figura 3.2, en dicha interfaz se pueden distinguir varios elementos:

En primer lugar se dispone de un editor de codigo para programar en lenguaje de programación **Arduino**, basado en C++, donde, como se puede observar, se distinguen tres partes diferenciadas. Al principio, previo a *void setup()*, habrá espacio para poder introducir las librerías que se importarán y la declaración de las variables que se utilizarán a lo largo del programa.

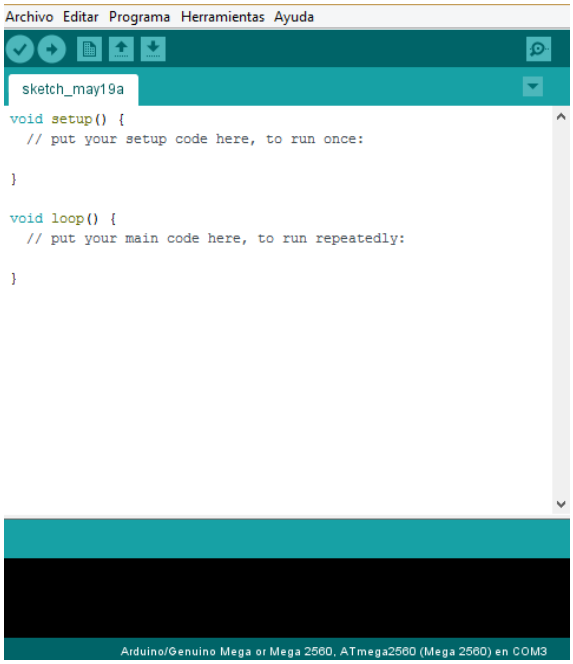


Figura 3.2 Interfaz del programa Arduino

Posteriormente se encuentra la funcion *void setup()* que se ejecutará una única vez al principio del programa y una funcion *void loop()* que se ejecutará constantemente una vez haya finalizado la anterior.

Como se observa en la figura anterior, el enorno de programación también dispone de un depurador, un compilador y no menos importante, una interfaz gráfica de usuario (GUI), que permite al usuario interactuar con esta y enviar y recibir datos al Arduino a través del puerto serie. Dicho programa trabaja con archivos .ino.

Para este TFG en concreto se configurará el programa para poder ejecutar la placa anterior correctamente. Desde *Herramientas > Placa* seleccionaremos nuestra placa a utilizar: *Arduino/Genuino Mega or Mega 2560* y desde *Herramientas > Procesador* seleccionaremos el *ATMega 2560*. Por último, siempre deberá establecerse la velocidad a la que el ordenador deberá comunicarse con nuestro Arduino, esto deberá de realizarse en el *void setup()* a través de la línea de código *Serial.print(9600)*. Este valor está en *baudios* [41], que es una unidad de medida que se refiere a número de símbolos transmitidos por segundo, por lo que no debe confundirse con los *bps*. Si cada elemento transporta un bit entonces los baudios tambien se correspondrán con los *bps*.

Una vez realizado esto la placa estará lista para interactuar con Arduino.

En primer lugar, se necesita una manera para comunicar el Smartphone con Arduino. Esta conexión se hace a través de un módulo *Bluetooth HC-05 ZS-040* como el que se muestra en la figura 3.3 de la marca TECNOIOT.

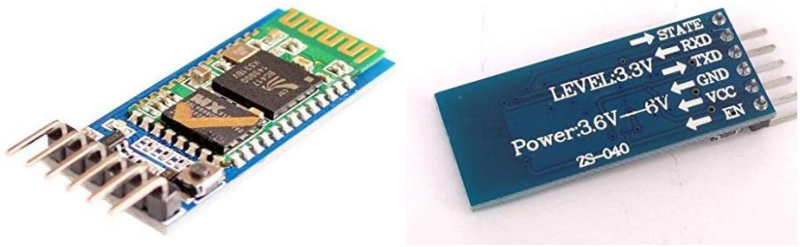


Figura 3.3 Modulo Bluetooth "HC-05 ZS-040" [42]

Como se ha visto en el capitulo de Estado del Arte, en una comunicación un maestro es quien inicia el emparejamiento y quien puede disponer de varios "esclavos". En este caso, el Smartphone será el maestro y el módulo *Bluetooth* será el esclavo.

Nuestro módulo *HC*-05 ya viene establecido por defecto como esclavo, pero se analizará cómo poder visualizar esto y como acceder a la configuración de dicho módulo. Para definir la configuración del módulo primero hay que definir la velocidad a la que deberá funcionar dicho dispositivo. La velocidad por defecto para funcionar en el modo de configuración es de 38400 *baudios*.

El esquema de conexionado para ésto se muestra en la figura 3.4, realizado con el programa *Fritzing* [43]. En otros modelos de *HC-05* es posible que no haga falta introducir ninguna resistencia, pero para el modelo *ZS-040* se necesita introducir un divisor de tension que reduzca a 3.3 V los 5 V de Arduino. Esto se conseguirá mediante dos resistencias, una de valor 1KΩ y otra de valor 2KΩ (Dos resistencias de 1K en serie)

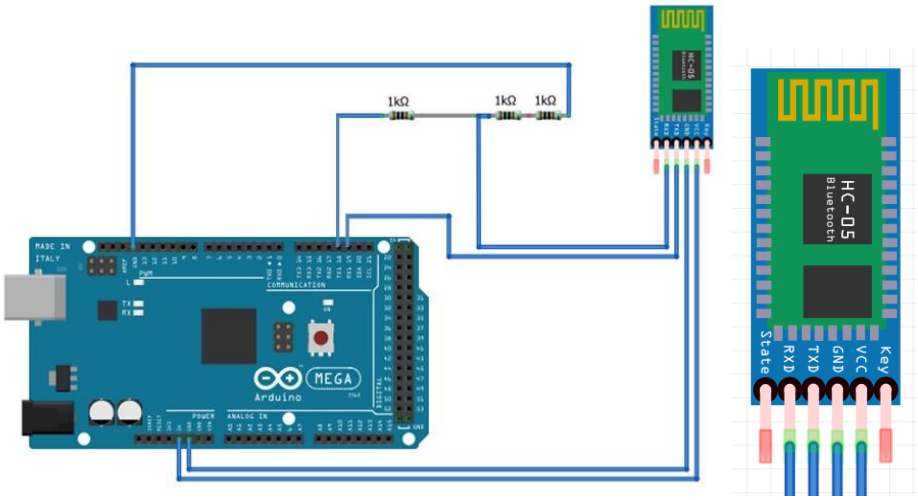


Figura 3.4 Esquema de conexionado para configurar el HC-05

Para poder entrar en el modo de configuración hay que, o bien suministrar 3.3 V al pin 34 del módulo *Bluetooth*, indicado en la figura 3.5, o bien presionar el pequeño botón switch que dispone antes de conectarlo a la alimentación.

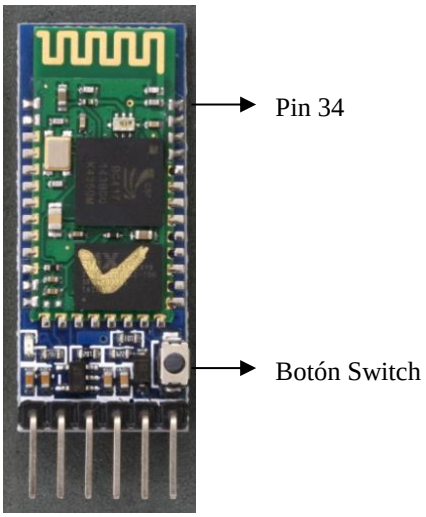


Figura 3.5 Entrar al modo configuracion con el "HC-05" [44]

Se ha creado un pequeño programa para acceder a la configuración del módulo el cual puede apreciarse en la figura 3.6. Como se puede observar, primero se definen los pines, que tendrán el objetivo de enviar y recibir datos. Los pines de las placas Arduino que comiencen por *TX* serán los que la placa utilice para enviar datos, mientras que los que comiencen por *RX* se utilizarán para recibirlos.

En este caso se ha decido utilizar los pines 18 y 19 de comunicación en la placa Arduino, que se corresponden con el *TX1* y *RX1* respectivamente. Éstos se identificarán en el programa a traves del comando *Serial1* automáticamente. Se ha preferido no utilizar los propios puertos serie *TX0* y *RX0* del Arduino ya que son los que utiliza para comunicarse con el ordenador vía USB.

Figura 3.6 Programa para configurar el modulo "HC-05 ZS-040"

A continuación se ha de declarar la velocidad de comunicación para el módulo *Bluetooth*, que, como se ha comentado anteriormente debe ser de 38400 *baudios* para el modo de configuración y la velocidad *Serial* ha de igualarse en baudios para poder trabajar con el monitor Serial y que el Arduino funcione correctamente. Antes de continuar con la linea *BTSerial.print("AT\r\n");* se aclarará el funcionamiento de dicho módulo en modo de configuración.

El módulo *HC-05* trabaja con comandos *AT*, es decir, a través de la comunicación serie *TX/RX* y el monitor Serie del que dispone Arduino se podrá cambiar la configuración de este utilizando comandos *AT*. Cuando se introduzcan los comandos se deberá introducir un *\r\n* ya que el módulo espera a que la línea acabe en dicha combinación, esto se puede indicar facilmente desde la pantalla del monitor serie que nos ofrece Arduino.

Para ello se debe modificar el campo que se encuentra a la izquierda de la velocidad de comunicación y seleccionar *"Ambos NL & CR"*, esto permitirá que al introducirse los comandos se agrege automáticamente al final de la línea y el módulo pueda detectar la introducción de este. De esta manera se recibirá un

"OK" en nuestro monitor serie, indicando que este se encuentra en disposición de recibir comandos y en modo de configuración.

En la funcion *void loop()*, que estará ejecutandose continuamente, se programa que, si el módulo *Bluetooth* tiene algo que devolver, se imprima en nuestro monitor serie y viceversa, si tenemos que enviarle un comando, que este lo reciba y permita enviar una respuesta. A continuación se muestran los comandos *AT* más comunes que pueden enviarse al módulo.

COMANDO	DESCRIPCIÓN
AT\r\n	Prueba la conexión, debe responder con "OK"
AT+ROLE=1	Configura el módulo en modo maestro
AT+ROLE=0	Configura el módulo en modo esclavo
AT+VERSION?	Nos devuelve la version del firmware
AT+BAUD<NUM>	Establece la velocidad en baudios a la que trabajará
AT+NAME<NOMBRE>	Establece el nombre del módulo
AT+PSWD<PIN4DIGIT>	Establece el PIN para emparejarse al módulo
AT+NAME?	Devuelve el nombre del módulo
AT+PSWD?	Devuelve la contraseña del módulo
AT+UART<BAUD><STOPBIT><PARITY>	Establece la velocidad de comunicación

Tabla 3.1 Comandos AT para el módulo HC-05

Nuestro módulo se ha actualizado a traves del comando *AT+NAME* al nombre de *VIBT*, el cual vincularemos desde nuestro Smartphone. En este apartado cabe destacar que, a lo largo del TFG, ha surgido un problema con el envío y recepción de datos, ya que en un principio se interpretó que la conexión *Android-Bluetooth* trabajaría correctamente a 9600 *baudios*, pero habia envíos y recepciones que tardaban o se enviaban de manera incorrecta. Tras búsquedas exhaustivas se consiguió la información de que los *Smartphones* actuales trabajan con el *Bluetooth* a una velocidad de 115200 *baudios* por lo que la velocidad de comunicación del *HC-05* ha sido modificada a dicho valor mediante el comando *AT+UART*=115200, 0, 0. El primer argumento se corresponde con la velocidad de comunicación, el segundo con el bit de parada, que podrá ser 0 si se trabaja con 1 bit o 1 si se trabaja con 2 bits y por último, el bit de paridad, que podrá ser 0, 1 o 2 según el tipo de paridad.

Para realizar las medidas de la temperatura, humedad e índice de calor será necesario disponer de un módulo *DHT11*, figura 3.7, que proporcionará éstos valores y se enviarán a nuestro módulo *HC-05*.

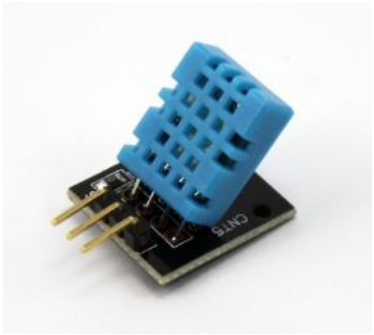


Figura 3.7 Sensor de temperatura y humedad DHT11 [45]

Como se puede observar en la imagen, el sensor dispone de tres pines, uno para alimentación, otro para tierra y otro para el envío de datos.

Los pines de tierra y alimentación se conectarán a sus correspondientes líneas, mientras que el pin de envío de datos se conectará a uno de los pines digitales que dispone nuestro Arduino, en nuestro caso, se conectará al pin 47.

Para el correcto funcionamiento del módulo debe instalarse previamente las librerías correspondientes. En nuestro caso se han instalado las librerías *DHT.h* y *Adafruit_Sensor.h* proporcionadas por la empresa *Adafruit*[47].

Para el control de los motores se ha utilizado una placa llamada Motor Driver Shield, que dispone de dos L293D para controlar los motores, actuando como puentes H y un 74HC595, encargado de controlar 8 pines digitales (4 para cada L293D) a partir de 3 entradas digitales del Arduino.

A continuación se muestra la placa utilizada en el proyecto con sus elementos:

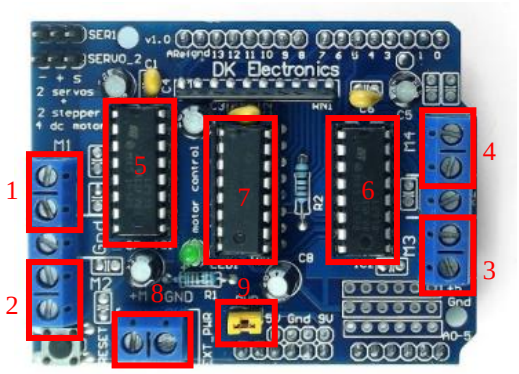


Figura 3.8 Placa Motor Driver Shield [46]

Los elementos 1 y 2 corresponden a los motores M1 y M2 controlados por el L293D enumerado como 5; los motores M3 y M4 se controlan con el del elemento 6. El elemento 7 se corresponde con el integrado 74HC595 y el 8 con la alimentación de la placa. Esta alimentación puede hacerse de tres maneras; la primera, utilizando la misma que Arduino, a través del DC jack que dispone Arduino o a través de una fuente externa en el elemento 8 y dejando conectado el jumper amarillo del elemento 9 (de esta manera compartirían la misma alimentación); la segunda, alimentando la placa con una fuente externa en 8 y Arduino mediante USB, para esto se debería de quitar el jumper; o la tercera,

alimentando Arduino mediante el DC jack y la placa por alimentacion externa, jumper quitado.

Para evitar errores se ha utilizado el segundo método, asi se alimentan las dos por separado y eliminamos posibles ruidos en la conexión. En la Motor Shield utilizaremos una bateria formada por 6 pilas de 1.5V cada una, es decir, se alimentará con 9V, suficientes para suplir los requisitos de los motores.

El funcionamiento del código en Arduino y la conexión con nuestro *Smartphone* se analizará más adelante en el apartado 3.3

3.2 Aplicación móvil

Para el desarrollo de esta utilizaremos el dispositivo físico *Huawei P20 Lite*, que dispone de la versión 8.0 Oreo, una versión más actualizada pero que nos permitirá hacer uso sin problema de dicha aplicación. Se utilizará la herramienta *Android Studio*. La aplicación se ejecutará a la versión comentada anteriormente, **API 21 5.0 Lollipop** y esto lo marcaremos al inicio de la aplicación.

Este programa trabaja con cuatro bloques importantes llamados *Activity* (actividad), *Broadcast Intent Reciever* (receptor de emisiones de intentos), *Service* (servicio) y *Content Provider* (proveedor de contenido). Para que el trabajo no se extienda innecesariamente se concretarán y definirán cada uno en caso de que se utilicen.

En primer lugar, se trabajará con *Activities*. Una *Activity* es el bloque más común utilizado en *Android Studio*, es una ventana o un cuadro de diálogo en una aplicación de escritorio. La aplicación podrá ir moviéndose entre las diferentes *Activities* de manera programada, ya que son entidades independientes que son capaces de llamarse entre ellas, compartiendo información o haciendo lo que el programador desee. Cada *Activity* dispondrá de un archivo .java y un archivo .xml, este último servirá para diseñar la interfaz que se visualizará, donde el

programador podrá incluir cualquier elemento deseado y dar sentido y forma a cada una de las características de estos. El archivo .java supondrá la parte de programación de dicha *Activity,* como por ejemplo la respuesta al pulsar determinado botón, incluido en la interfaz anterior, o las tareas a realizar cuando dicha *Activity* se inicie.

Como se ha comentado anteriormente en una *Activity* se podrán incluir múltiples funciones y bloques (*Views*) para generar la interfaz de usuario, como imágenes, botones, *Widgets* y otros incluidos en la figura 3.9

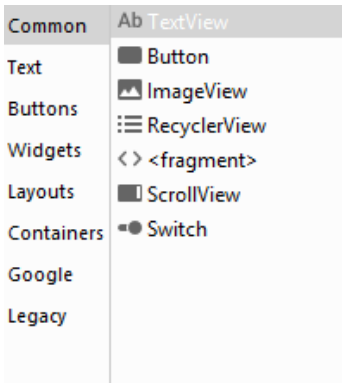


Figura 3.9 Views posibles en la interfaz de una Activity

La actividad principal de nuestra aplicación se llamará "MainActivity", esta constituirá la portada de la aplicación. En ella se nos mostrará un sencillo botón (*Button*) de bienvenida que conducirá a la pantalla de conexión.

En los archivos **.xml**, que se pueden encontrar en la pestaña *res>layout*, se podrá visualizar dos pestañas principales, una de *Design* (Diseño) y otra de *Text* (Texto). La primera pestaña servirá para visualizar la interfaz de manera intuitiva, mostrando el menú indicado en la figura anterior donde se podrá introducir las características de cada elemento introducido y el diseño del Smartphone, aquí se podrá cambiar el modelo de móvil y por ende el tamaño de pantalla sobre el que se trabajará. Es un sistema de edición visual sencillo; en la figura 3.10 se puede observar esta interfaz, donde, en la zona derecha se dispone de un conjunto de *Attributes* (Atributos) que caracterizan al elemento que se seleccione, pudiendo modificar cualquier valor a nuestro gusto.

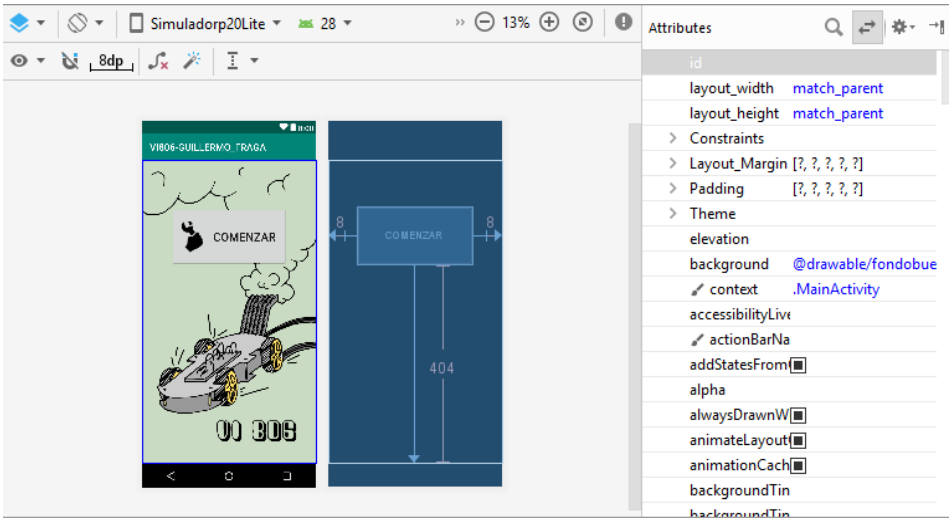


Figura 3.10 Pestaña "Design" del archivo xml

La segunda pestaña se corresponde con el xml propiamente dicho, donde se pueden establecer las características de cada elemento tal y como se puede hacer desde la interfaz gráfica *Design* pero de manera textual y programada. Más adelante se explicará esta parte de más detalladamente.

Para este trabajo en concreto se ha creado un simulador con las características del Smartphone donde se probará la aplicación, que como se ha comentado anteriormente es el *Huawei P20 Lite*, estas características se muestran en la tabla 3.2

Type	Name	Play Store	Resolution	API	Target	CPU/ABI
	Simuladorp20Lite API 23		1080 × 2244: xxhdpi	23	Android 6.0 (Google APIs)	x86

Tabla 3.2 Características de pantalla del dispositivo utilizado

La pantalla de bienvenida o portada se mostrará como se indica en la figura 3.11 El fondo ha sido creado en *Adobe Illustrator CC 2018* mediante escaneo y vectorización a partir de dibujos personales.

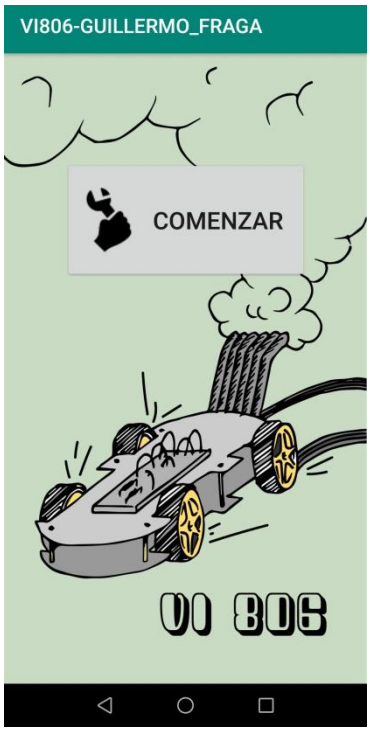


Figura 3.11 Interfaz de la actividad "MainActivity"

En la figura 3.12 se encuentran las características de la interfaz, la pestaña *Text* que se comentaba anteriormente, es decir, el archivo .xml que hace de la *Activity* nuestra portada.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/fondomain"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/bComenzar"
        android:layout_width="241dp"
        android:layout_height="120dp"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="404dp"
        android:drawableStart="@drawable/logomain"
        android:text="COMENZAR"
        android:textSize="24sp"
        android:textStyle="normal"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.504"
        app:layout_constraintStart_toStartOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Figura 3.12 Pestaña "text" del "MainActivity"

En dicha figura se muestran dos elementos, el *ConstraintLayout* y el *Button*. El primer elemento supone la pantalla donde se podrán introducir diseños, textos, botones…etc., y permite situarlos a cierta distancia de los bordes o de los otros elementos existentes, independientemente de la pantalla que se utilice. También se podría haber utilizado un *RelativeLayout* como mesa de trabajo, pero el *ConstraintLayout* tiene un manejo más sencillo, más completo y una de las grandes ventajas que presenta es que podemos trabajar sin apenas editar el .xml.

En el *ConstraintLayout* únicamente se destacará el a*ndroid:background* y el *android:layout_width/height=”match_parent”* donde se indica el ancho y el largo del elemento en cuestión. El *match_parent* indica que esta pantalla no tiene unos límites definidos, si no que se adaptará al tamaño de la pantalla que se utilice. En el *background* se introducirá el archivo realizado en *AI* (Adobe Illustrator) en formato .PNG llamado “fondomain”, que tendrá que ser guardado previamente en la carpeta *res>drawable*

El *Button* constituye el botón que se muestra en la figura 3.12 y se pueden observar características como el tamaño del texto en *android:textSize*, el tamaño del botón y la posición de sus márgenes en *marginStart/marginEnd/marginBottom*. El icono del interior del botón llamado “logomain” se inserta en formato .PNG en la carpeta *drawable* y posteriormente en donde indica *android:drawableStart*.

También hay que destacar el concepto de *id*, ya que cada elemento introducido en la actividad debe poseer uno, que lo hará diferente del resto y será el nombre que lo identifique a la hora de programar, en el caso del botón se le ha denominado “bComenzar”.

A continuación se detalla el archivo **.java** de nuestra actividad principal o portada, que se encuentra en el directorio *java>”Nombre del proyecto”* junto con los demás archivos de este formato y se muestra en la figura 3.13. Estos archivos se encargan de almacenar el código de la aplicación y las acciones que en esta se produzcan.

```
package com.example.asem.vi806_guillermo_fraga;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    private Button botoncomenzar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        botoncomenzar=(Button) findViewById(R.id.bComenzar);

        botoncomenzar.setOnClickListener((v) → {
                Intent pantprinc=new Intent( packageContext: MainActivity.this,Conectarbtactivity.class);
                startActivity(pantprinc);
                finish();
        });


    }
}
```

Figura 3.13 Archivo .java del "MainActivity" o portada

Para poder continuar la explicación de este archivo habrá que hacer una breve explicación de los *Intent* (Intento). Los *Intent* son mensajes que se lanzan para notificar desde eventos del propio sistema hasta eventos de nuestra propia aplicación. Se utilizan como comunicación entre elementos del dispositivo. Dentro de los mensajes del *Intent* se encuentran la acción a realizar y los datos que van a intervenir en dicha acción.

De la figura anterior, se observa que los archivos .java se componen de, en primer lugar, el paquete o carpeta al que pertenece, en este caso llamada "VI806_Guillermo_Fraga", de los *import* o librerías necesarias para la ejecución del código y de la clase que contiene, en este caso *MainActivity*.

Para dar sentido a la interfaz necesitamos establecer en el código las funciones que precisen los elementos que se encuentren en esta.

Previamente se declaran los elementos existentes en nuestra interfaz (*private botoncomenzar*) y después el código que se explicará a continuación, pero antes ha de aclararse el funcionamiento de las *Activity*. En la figura 3.14 se muestra el

ciclo de vida de una *Activity* y cómo va llamando a diferentes métodos a lo largo de su ejecución.

Como se puede observar en esta figura, tenemos cuatro posibles estados y siete posibles métodos.

Los cuatro estados posibles de una *Activity* son los siguientes:

- Activa:

  Una actividad se encuentra en Activa cuando esta se muestra en primer plano y el usuario puede interactuar directamente con ella, es decir, cuando se encuentra la primera en la pila de ejecución.

- Pausada:

  En este estado se encuentran las aplicaciones que están en segundo plano pero aún son visibles en la interfaz, es decir, cuando una actividad se coloca encima de otra pero sin taparla del todo, ya sea porque la interfaz de esta es menor que la anterior o porque es transparente.

- Parada:

  En caso contrario al estado de Pausa, aquí la actividad nueva pasa a superponerse por completo a la anterior, quedando esta completamente cubierta.

- Destruida:

  La actividad que se encuentra destruida ya no está disponible, si se quiere volver a ella se deberá crear de nuevo y comenzar un nuevo ciclo de vida

Los siete posibles métodos que pueden ejecutarse entre los diferentes estados son los siguientes:

- *onCreate()*:

  A este método se le llama nada más crear la actividad y es donde se prepara la interfaz gráfica de la pantalla, enlazando los datos con sus respectivos métodos de visualización.

- *onStart()*:

  Este método es llamado después del *onCreate()* o del *onRestart()* que se explicará más adelante. Esta se ejecuta justo antes de que una actividad pase a ser visible para el usuario.

- *onResume()*:

  Se ejecuta justo antes de que el usuario pueda interactuar con la actividad presentada.

- *onPause()*:

  Este método es llamado cuando se pasa a otra actividad diferente desde la actividad actual, es decir, cuando se llama al método *onRestart()* de la nueva actividad. Aquí se ha de liberar todo lo posible que consuma recursos ya que hasta que no se haya ejecutado el método en su totalidad no pasará al siguiente método y la aplicación se retrasará.

- *onStop()*:

  Se ejecuta cuando la actividad pasa a ser imperceptible por el usuario, ya sea porque otra actividad se ha situado en primer plano y ha tapado por completo a la anterior o porque simplemente la actividad ha sido destruida y se llamará al método *onDestroy()*.

- *onDestroy()*:

  A este método se llama antes de destruir una actividad, perdiendo todos los datos asociados a dicha *Activity*, por lo que aquí es donde se deberá de controlar la persistencia de datos.

- *onRestart()*:

  Este método se ejecuta cuando una actividad vuelve a activarse después de haber estado parada, justo antes de que comience de nuevo.

En la figura anterior, en el archivo .java se puede ver como la clase llama al método *onCreate* al ejecutarse y este muestra el *layout* (interfaz) del archivo .xml.

Siempre que se implemente una llamada a un método de los nombrados anteriormente se deberá implementar también la llamada al correspondiente método de la superclase[4] tal y como se aprecia en la figura anterior en ***super.****onCreate(savedInstanceState)*.

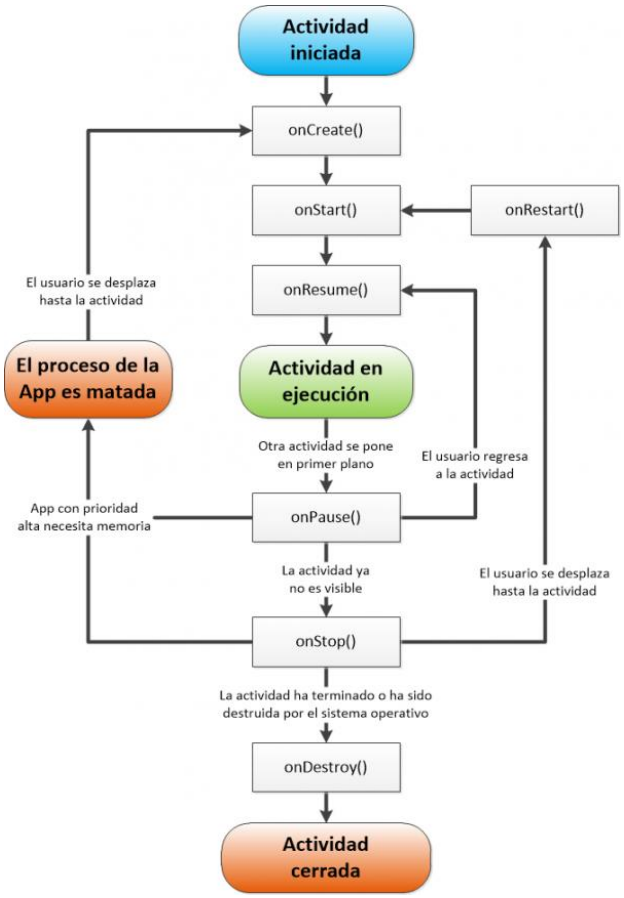


Figura 3.14 Ciclos de vida de una Activity en Android [48]

[4] Clase padre de la que se heredan los atributos y métodos

En la siguiente línea de código de la figura 2.13 se asocia el botón declarado con anterioridad al botón de la interfaz, con el *id* que le pertenece, en este caso "bComenzar".

Posteriormente, se programa la función *setOnClickListener()* y *onCLick()*, que se encargan de ejecutar los procesos que se encuentran en el interior de estas cuando el botón se ha pulsado y soltado. En ella, como en este caso únicamente se trata de una portada, se ha programado un *Intent* llamado "pantprinc" que se encargará de llevar al usuario desde donde se le indique hasta el objetivo, en nuestro caso desde la actividad *MainActivity* hasta *Conectarbtactivity*, que será la actividad que presente los dispositivos *Bluetooth* a los que nos podemos conectar.

En *Conectarbtactivity* se encontrará una interfaz donde se creará una lista mediante el elemento *ListView*, encargado de crear una lista con los elementos que el programador le indique y un *TextView*, esto se puede observar en la figura 3.15. En este caso en la lista se mostrarán los dispositivos que se encuentran vinculados a nuestro Smartphone a través de *Bluetooth* y su dirección *MAC*, una dirección única para cada dispositivo.

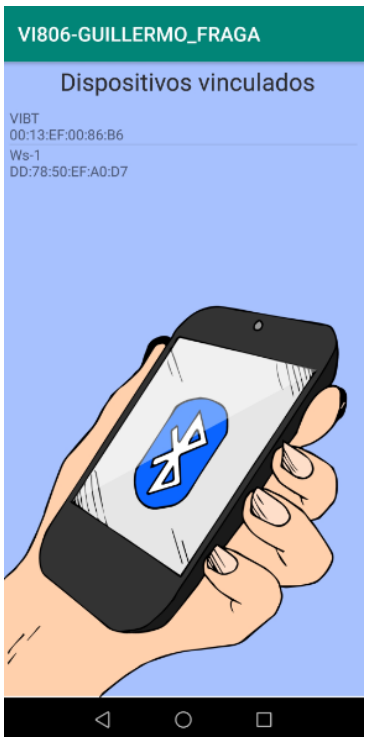


Figura 3.15 Interfaz gráfica de "Conectarbtactivity"

Para llevar a cabo esto se ha creado otro archivo *.xml* llamado "*nombres_dispositivos"* que servirá como *layout* o espacio para mostrar el *Array*[5] de cadenas de caracteres o *Strings* que contendrán el nombre de los dispositivos enlazados y su dirección *MAC*, ligando de esta manera el *ListView* de la actividad con el *.xml*.

Previamente se ha de dar los permisos de acceso necesarios para la utilización del *Bluetooth*. Estos permisos se deberán solicitar a través del archivo de texto *.xml* de *Android Manifest* que se encuentra dentro de la carpeta *manifests* > *AndroidManifests.xml* como se muestra en la figura 3.16

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

Figura 3.16 Permisos de acceso Bluetooth

En el archivo *.java* se encuentra la programación de esta *Activity*. Esta vez se elaborará en la función *onResume()* ya que la función *onCreate()* solo se ejecutará una vez al crearse dicha actividad, y en caso de necesitar conectarse a otro dispositivo no podrá hacerse desde dentro de la aplicación, por lo que se ha programado en dicho método. Esta función se aprecia en la figura 3.17.

```
@Override
public void onResume(){
    super.onResume();
    Listabt=(ListView) findViewById(R.id.IdListavincul);

    ComprobarBT();

    ArrayDispostivincul = new ArrayAdapter( context: this, R.layout.nombres_dispositivos);
    Listabt.setAdapter(ArrayDispostivincul);
    Listabt.setOnItemClickListener(mDeviceClickListener);

    AdaptadorBT=BluetoothAdapter.getDefaultAdapter();

    Set <BluetoothDevice> Vinculados =AdaptadorBT.getBondedDevices();
    if((Vinculados).size()>0){
        for (BluetoothDevice device : Vinculados){
            ArrayDispostivincul.add(device.getName() + "\n" + device.getAddress());
        }
    }
    else ArrayDispostivincul.add("No hay dispositivos vinculados");
}
```

Figura 3.17 Función onResume() de la actividad "Conectarbtactivity"

[5] Vector de almacenamiento de elementos del mismo formato

En primer lugar se ejecuta la función "*ComprobarBT()"*, disponible en la figura 3.19, que es una función que detecta si el dispositivo dispone de *Bluetooth*. Se alamacena en el *BluetoothAdapter* definido como *AdaptadorBT*, el adaptador local *Bluetooth* del Smartphone, en caso de que no disponga nos avisa y en caso de que si disponga pero no esté activado se lanza un *Intent* que advierte al usuario de que una aplicación desea activar el *Bluetooth*, para que este pueda permitirlo:

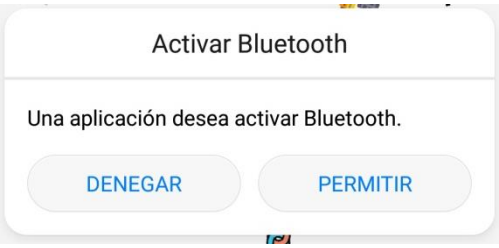


Figura 3.18 Ventana activación Bluetooth

Esto se hará a través de métodos como *.isEnabled()*, *AdaptadorBT=null* y *ACTION_REQUEST_ENABLE*.

```
private void ComprobarBT(){
    AdaptadorBT=BluetoothAdapter.getDefaultAdapter();
    if(AdaptadorBT==null){
        Toast.makeText(getBaseContext(), text: "El dispositivo no tiene Bluetooth", Toast.LENGTH_SHORT).show();
    }
    else{
        if(AdaptadorBT.isEnabled()){  //Detecta si tenemos el BT activado
            Toast.makeText(getBaseContext(), text: "Bluetooth activado", Toast.LENGTH_SHORT).show();
        }
        else{
            Intent activarBtIntento=new Intent (BluetoothAdapter.ACTION_REQUEST_ENABLE); //Lanza la peticion de activar bluetooth
            startActivityForResult(activarBtIntento, requestCode: 1);
        }
    }
}
```

Figura 3.19 Función ComprobarBT() de la actividad "Conectarbtactivity"

En esta función se utilizan notificaciones para dirigirse al usuario a partir de elementos *Toast*, los cuales emiten un mensaje que se reproduce en pantalla durante un tiempo determinado.

Se puede observar en las líneas de código como en primer lugar se determina el mensaje a mostrar y posteriormente la duración de este, que podrá ser *LENGHT_SHORT* o *LENGTH_LONG* dependiendo del objetivo, seguido de un *.show* para mostrarlo por pantalla.

En la siguiente línea de la función *onResume()* se vincula, como se ha comentado anteriormente, el *layout* con el *.xml* de "*nombres_dispositivos"* para formar el *Array* de contenido, y dicho *Array* se vincula a su vez con la lista creada en la interfaz. De esta manera se podrá visualizar la lista de dispositivos vinculados con nuestro Smartphone a través de *Bluetooth*.

Esta información se proporciona a través de *getBondedDevices*, que retorna los objetos *BluetoothDevice*, es decir, los dispositivos Bluetooth que están enlazados con el adaptador local, se guardan en *"Vinculados"* y mediante el método *.size()* se obtiene el número de dispositivos vinculados a nuestro Smartphone.

Posteriormente se realiza la estructura de control *for* tantas veces como dispositivos haya detectado el *.size*, y se mostrará en la interfaz tanto el nombre como la dirección MAC de éstos.

Tras obtener un listado con los dispositivos enlazados se procede a ejecutar la función *OnItemClickListener()*, que se encarga de ejecutar una acción cuando el usuario pulsa alguno de los dispositivos que se encuentran dicha lista. Esta función se puede observar en la figura 3.20.

```
private AdapterView.OnItemClickListener mDeviceClickListener = new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String info= ((TextView) view).getText().toString();
        String address= info.substring(info.length() - 17);

        Intent volvermenu = new Intent( packageContext: Conectarbtactivity.this, Menuppalactivity.class);
        volvermenu.putExtra(EXTRA_DEV, address);
        startActivity(volvermenu);
        finish();
    }
};
```

Figura 3.20 Función OnItemClickListener() de "Conectarbtactivity"

El objetivo de esta actividad es conseguir la dirección MAC a la que debemos conectar nuestro Smartphone, y enviar este dato a la siguiente actividad. Como

las direcciones MAC están compuestas de 6 bloques de dos caracteres separados por dos puntos (17 caracteres en total), se cogen los últimos 17 caracteres del *string* utilizando métodos como *lenght()* y *substring()*, que almacena los caracteres desde una posición a otra posición concretas. Una vez conseguida dicha información se lanza un *Intent* que envía la dirección a la siguiente actividad, llamada "*Menuppalactivity*".

En esta se programará el menú, en el que habrá una serie de funciones disponibles, que a lo largo de este TFG se irán programando y añadiendo, quedando todas aquí indicadas y explicadas. El menú debe ser una interfaz sencilla y clara, ya que como se indicó en la introducción del documento presente, la aplicación ha de ser útil para cualquier usuario, independientemente de si es una persona joven como una persona de tercera edad sin conocimientos.

El menú principal estará formado por botones (*Button*), los cuales ya se han explicado con anterioridad y por elementos *TextView*, que son únicamente cuadros de texto, ya que esta actividad será un paso intermedio entre las diferentes funcionalidades de las que dispondrá nuestro programa.

Para ello se almacenará en primer lugar la dirección MAC obtenida desde la actividad anterior. Se ha decidido destruir en vez de pausar las actividades inútiles para liberar memoria, como pueden ser la portada o la actividad anterior *"Conectarbtactivity"*.

Figura 3.21 Interfaz gráfica de "Menuppalactivity"

El archivo *.java* de *"Menuppalactivity"* se muestra en la figura 3.22, en él únicamente se destacará el método para obtener la dirección MAC ya que los otros elementos presentes ya se han visto anteriormente.

A través de la función *getIntent()* y *getStringExtra()* se obtiene el dato que se envía a través del *Intent* que abre la actividad. Este se almacena en la variable *dirmac*, declarada como *String*, que posteriormente enviaremos a las diferentes funciones del menú para enviar o recibir datos del módulo *Bluetooth*.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_menuppalactivity);

    botonestado=(Button) findViewById(R.id.bEstado);
    botoncontrolar=(Button) findViewById(R.id.bControlar);

}
@Override
protected void onResume(){

    super.onResume();
    Intent MAC = getIntent();
    dirmac=MAC.getStringExtra(Conectarbtactivity.EXTRA_DEV);

    botoncontrolar.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent controlar = new Intent( packageContext: Menuppalactivity.this,Controlarvehiculoact.class);
            controlar.putExtra(Conectarbtactivity.EXTRA_DEV,dirmac);
            startActivity(controlar);
        }
    });

    botonestado.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent pantconect=new Intent( packageContext: Menuppalactivity.this,Conectarbtactivity.class);
            pantconect.putExtra(Conectarbtactivity.EXTRA_DEV,dirmac);
            startActivity(pantconect);
        }
```

Figura 3.22 Funciones onCreate() y onResume() de "Menuppalactivity"

3.3 Intercambio de datos

En este apartado se analizarán los procesos que intervienen en la conexión entre Smartphone y el *Bluetooth*, es decir, entre los dos apartados anteriores.

En la conexión *Bluetooth*, al igual que en otros protocolos, intervienen tres elementos: El servidor, el cliente y la conexión.

El servidor se encarga de intervenir en las conexiones entrantes, aceptarlas y establecer la conexión, mientras que el cliente solicita la conexión y la establece una vez el servidor la haya aceptado. Esto implica que el hilo de la conexión será lanzando tanto por el cliente como por el servidor.

Para llevar esta conexión, se intercambian datos a través de *Sockets*, que son canales o métodos de comunicación entre cliente y servidor en este caso. Cada interlocutor crea un *socket*, dando lugar de esta manera a un canal de doble dirección.

En la siguiente figura se podrá observar el modo de funcionamiento entre servidor y cliente a través de *sockets*.

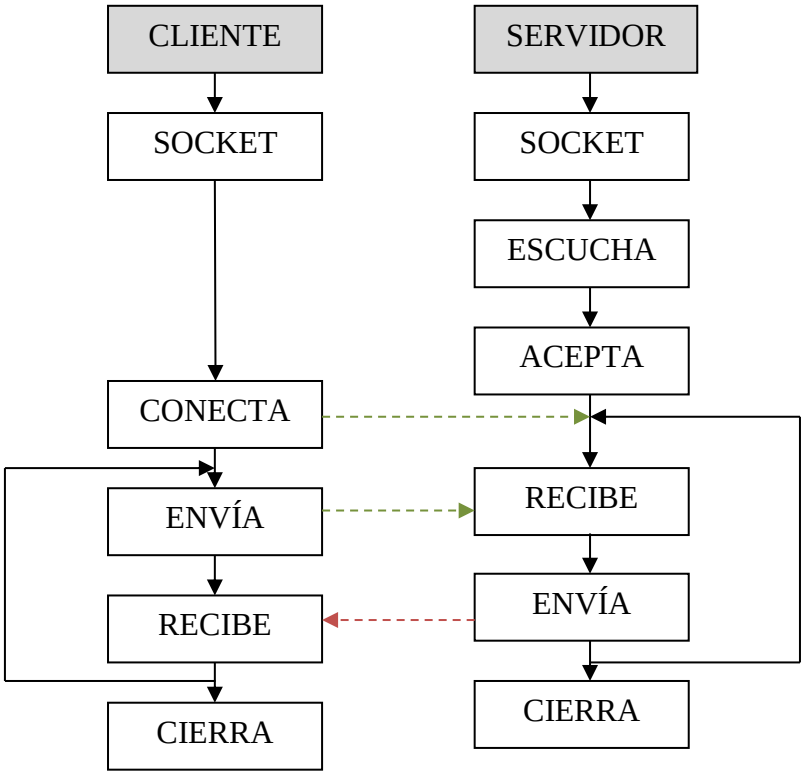


Figura 3.23 Modo funcionamiento Cliente-Servidor

Para llevar a cabo esta conexión se ha de incluir dos clases que se verán posteriormente en el código de manera más detallada. La primera clase que se ha de incorporar es la clase *Thread*, que permite a la aplicación realizar varias tareas de manera simultánea actuando como hilo, tal y como su nombre indica. La otra clase necesaria será la llamada *Handler*, esta permite a la aplicación o actividad administrar y procesar los distintos mensajes entre los diferentes *Threads* y así puedan enviar mensajes a nuestro hilo principal de la aplicación.

Todas estas aplicaciones se irán explicando más detalladamente a medida que se analicen las diferentes actividades.

En nuestro menú principal se programarán varias actividades que realizarán diferentes funciones. Estas funciones se muestran en los siguientes apartados:

3.3.1 Estado del vehículo

El objetivo de esta actividad será simular y manejar la obtención de datos interesantes del vehículo para que el usuario permanezca informado acerca de la situación en la que se encuentra este.

En esta función se podrá obtener datos como la temperatura en el interior del vehículo, la humedad, el índice de calor y la autonomía del vehículo.

La autonomía del vehículo se obtendrá a partir del nivel de vida de la batería que incorpore. Al tratarse de un prototipo, el objetivo es conseguir simular una medición acerca de la autonomía en un vehículo real, no se podrá trabajar con un combustible real por lo que la batería suplantará a este.

En este apartado no se ha podido incluir por falta de tiempo una posible reacción del vehículo ante una decisión del usuario sobre la temperatura, como podría ser añadir una resistencia que según la temperatura del interior del vehículo o según le indique el usuario se calentase o, a través de un ventilador, se enfriara.

Para llevar a cabo esta función ha sido necesaria la siguiente programación:

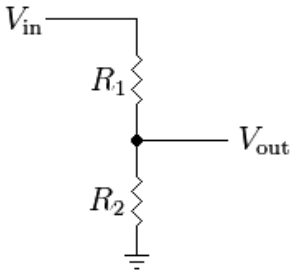
```
if(opcion=='1'){
  int volt=analogRead(8);
  int nivel=0;
  float volts = (volt/1023.0)*4.75;
  if(volts<5 && volts>=4.2){
    nivel=1;
  }
  else if(volts<4.2 && volts>=3.4){
    nivel=2;
  }
  else if(volts<3.4 && volts>=2.6){
    nivel=3;
  }
  else if(volts<2.6 && volts>=1.8){
    nivel=4;
  }
  else if(volts<1.8 && volts>0){
    nivel=5;
  }
  String nivelfin=String(nivel);
  String total;
  float humindic=dht.readHumidity();
  float tempindic=dht.readTemperature();
  float indicalor=dht.computeHeatIndex(tempindic,humindic,false);
  String temp=String(tempindic);
  String hum=String(humindic);
  String indical=String(indicalor);
  total=(temp+"#"+hum+"#"+indical+"#"+nivelfin);
  Serial1.print(total);
```

Figura 3.24 Lectura y escritura de datos del sensor DHT11 y autonomía

La primera variable (“opcion”), es un carácter en el que se almacenará el dato recibido desde el *Smartphone*. Para cada *Activity* que precise de una repercusión en el Arduino tendrá asociado un carácter que la definirá. En este caso, la actividad *Estadovehicactivity*, como se observará más adelante, se enviará el carácter ‘1’ para que en Arduino se puedan distinguir las acciones que se quieren llevar a cabo.

La función *void loop()* se ejecutará constantemente, por lo que el programa estará comprobando continuamente mediante la linea *if (Serial.available()>0)* si se dispone de informacion en el serial para intercambiar.

Para obtener la autonomía del vehículo ha sido necesario, en primer lugar, reducir los 9V de la alimentación de la Motor Shield a 5V. Esto es debido a que los pines analogicos de la placa Arduino Mega 2650 únicamente son capaces de leer valores entre 0 y 5V. Para ello se ha realizado un divisor de tensión con dos resistencias de 330 Ω, reduciendo así el voltaje a 4.5 V. Para este cálculo se ha utilizado la siguiente formula:



$$Vout = \frac{R_2}{R_1 + R_2} \cdot Vin$$

Figura 3.25 Divisor de tensión

Como se observa en la figura 3.24 la salida *Vout* se ha conectado al pin A8 del Arduino para recibir el voltaje. En Arduino los pines analógicos toman valores de 0 a 5V, y los transforman a valores enteros entre 0 y 1023 unidades, por ello, es necesario realizar una conversión previa para tratar los valores en unidades de voltaje. Posteriormente se indica el nivel de batería que se dispone; a través de

else-if anidados establecemos unos niveles de autonomía, de 1 a 5, siendo 1 autonomía al maximo y 5 autonomía al minimo.

Los valores de humedad, temperatura e indice de calor se reciben posteriormente mediante *float* y se convierten a String para enviar los datos. Una vez tenemos todos los datos en forma de String, se envían al modulo Bluetooth separados por un asterisco, para poder almacenar en Android Studio las diferentes variables e imprimirlas por pantalla.

Este dato se recibe mediante la actividad *Estadovehicactivity*, donde se imprimirán posteriormente por pantalla. Para esta recepción, se utiliza la función run del ConnectedThread (hilo de conexión), presentada en la siguiente figura:

```
public void run(){
    byte[] datos = new byte[256];
    int longitud;
    int x=0;
    String cadena="";
    while (true){

        try{
            longitud=mmInStream.read(datos);
            String readMessage = new String(datos,  offset: 0 ,longitud);
            cadena=cadena+readMessage;
            Log.d(TAG, cadena);
            if (x==1){
                bluetoothIn.obtainMessage( what: 0,longitud, arg2: -1,cadena).sendToTarget();
            }
            x++;
            if(x>1){
                x=0;
                cadena="";
            }
        } catch (IOException e) {
            break;
        }
    }

}
```

Figura 3.26 Recepción de datos en Android Studio

En esta se leen los bytes del buffer de entrada y se almacenan en forma de String. Debido a que nuestra cadena de datos se envía dividida es necesario almacenar la primera parte en la variable cadena y posteriormente añadir la restante en la misma, así, cuando dispongamos de la cadena completa la enviaremos a través del Handler. En la siguiente figura, para no extendernos, únicamente se muestra la recepción de datos a través de este:

```
public void handleMessage(Message mensaje) {
    Log.d(TAG, msg: "Entramos en handler");
    if (mensaje.what == 0) {
        String readMessage = (String) mensaje.obj;
        int x=0;
        char niv;
        String temp="",hum="",indical="",nivel="";
        for(int i=0; i<readMessage.length();i++){
            if(readMessage.charAt(i) != '#' && x==0){
                temp=temp+readMessage.charAt(i);
            }
```

Figura 3.27 Almacenaje de datos en el Handler

Como se puede observar, el mensaje se recibe en forma de objeto y se almacenan las variables temperatura, humedad, índice de calor y nivel, separadas por asteriscos. Una vez se tienen las variables almacenadas, se imprimen por pantalla a través de un TextView:

Figura 3.28 Interfaz gráfica de "Estadovehicactivity"

3.3.2 Control del vehículo

En la opción de controlar el vehículo el usuario podrá manejar el vehículo a distancia a través de una interfaz sencilla y clásica.

Esta función se llevará a cabo a través del envío de datos a nuestro módulo *Bluetooth* desde la actividad llamada *Controlarvehiculoact*. A continuación se puede observar la interfaz gráfica de esta actividad.

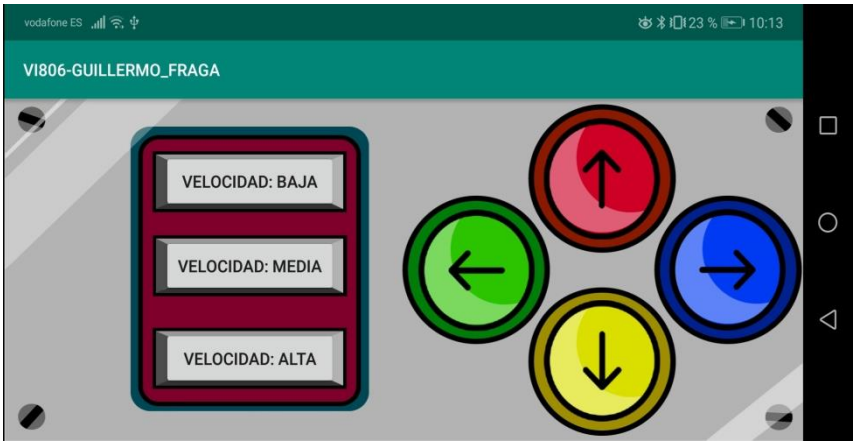


Figura 3.29 Interfaz gráfica de "Controlarvehiculoact"

Como se puede observar en la figura anterior, para este apartado se ha tomado la decisión de realizar la actividad en formato apaisado, para una mayor comodidad a la hora de controlar este.

Este cambio ha de realizarse en el *AndroidManifest.xml*. Como se indica en la figura 3.30, se ha optado por un apaisado que se adapta a la orientación del Smartphone a través de *sensorLandscape*, para dar también más facilidades al usuario.

```
<activity
    android:name=".Controlarvehiculoact"
    android:screenOrientation="sensorLandscape" />
```

Figura 3.30 Orientacion apaisada en Android Studio

Como se puede observar en la interfaz de la figura 3.29, es posible establecer la velocidad de los motores previamente a controlar el vehículo y no solo posible sino obligatorio, ya que así ha sido programado en nuestro Arduino, como se verá más adelante, debido a que antes de realizar un movimiento se debe indicar la velocidad de este para evitar posibles accidentes o reacciones inesperadas.

Los motores, dependiendo de la velocidad indicada recibirán voltaje en un rango de 3 a 6V, siendo 3 la velocidad mínima y 6 la velocidad máxima.

En el archivo *.java* se incluirán las siguientes librerías:

```
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;
```

Figura 3.31 Librerías para el control del vehículo y envío de datos

Las librerías aún no explicadas en este documento se explicarán en el momento de su utilización y el motivo de su introducción, en la programación que forma esta actividad. En la figura 3.32 se observan las variables necesarias para la conexión y control.

```
private BluetoothAdapter btAdapter=null;
private BluetoothSocket btSocket=null;
private static final UUID BTMODULEUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
public Controlarvehiculoact.ConnectedThread ConexionBT;
public static String macmac;
private static final String TAG="actividad controlar";

private Button barriba,babajo,bdcha,bizda,bbaja,bmedia,balta;
```

Figura 3.32 Variables para el control del vehículo y envío de datos

Para que el conexionado funcione correctamente se deberá, en primer lugar, establecer el adaptador *Bluetooth* del dispositivo local para poder realizar tareas fundamentales relacionadas con este, al cual pasaremos la dirección MAC de nuestro módulo. En segundo lugar crear un *BluetoothSocket* para poder administrar la conexión, posteriormente, un *UUID* (Identificador Único Universal), donde, a través de la librería *java.util.UUID* estableceremos una variable llamada *BTMODULEUUID* que poseerá el *UUID* indicado en la figura anterior, que es el que define al módulo *HC-05*. Este identificador está formado por 32 dígitos hexadecimales ordenados con formato 8-4-4-4-12 y separados por guiones. También será necesario el objeto *ConexionBT* de la clase ConnectedThread que será al que posteriormente se le asignará el socket, permitiendo crear un hilo de conexión. Esta clase es la que permite realizar el envío y recepción de datos, dispone de dos métodos, *run()* y *write()*, que se verán con detalle más adelante y permiten enviar o recibir datos. Por último una variable *String* donde se almacenará la dirección MAC de nuestro módulo, obtenida a través de *Intent*.

En la función *onCreate()* de la actividad únicamente se relacionan los botones con sus respectivos *Id* y se obtiene y almacena la dirección MAC a través de *getIntent()* y *getStringExtra()*, que ya se explicaron en la actividad del menú principal.

La función *onResume()* requiere más detenimiento, en ella se realiza la conexión, se inicia el *socket* y se declara el efecto de los *Button* incluidos en la actividad:

```
protected void onResume(){
    super.onResume();

    btAdapter=BluetoothAdapter.getDefaultAdapter();
    BluetoothDevice device = btAdapter.getRemoteDevice(macmac);

    try{
        btSocket = createBluetoothSocket (device);
    } catch (IOException e){
        Toast.makeText(getBaseContext(),  text: "La creacion del Socket falló", Toast.LENGTH_LONG).show();
    }
    try{
        btSocket.connect();
    }catch (IOException e){
        try{
            btSocket.close();
        }catch (IOException e2){
        }
    }
    ConexionBT = new ConnectedThread(btSocket);
    ConexionBT.start();
    ConexionBT.write( input: "2");
    Log.d(TAG, msg: "pasamos por el start");
```

Figura 3.33 Función onResume() de la actividad Controlarvehiculoact

En primer lugar se inicializa el *btAdapter* y se almacena el adaptador *Bluetooth* local del Smartphone, al cual enviamos y relacionamos la dirección MAC obtenida anteriormente en el ListView.

A través de la excepción *try…catch* se consiguen realizar dos acciones, una cuando todo funciona con normalidad, ejecutándose el código del interior del *try*, y otra cuando se produce algún error y java lanza una excepción, ejecutándose el interior del *catch*.

Tras lanzar el *socket* y realizar la conexión a través del hilo o *Thread* con *.start()*, nuestro dispositivo se conecta con el módulo *HC-05* y el LED de este pasa a parpadear dos veces cada 2 segundos aproximadamente.

Al programa de Arduino se le ha de indicar cuál es la opción que se ha escogido desde el menú principal y esto se hace enviando el valor *2*, que será recibido por el módulo y enviado al Arduino, más tarde se observará en dicho programa la ejecución. A continuación se explicará el propósito de la línea de código Log.

Android posee una herramienta llamada *Logcat*, que se utiliza como línea de comandos para mostrar un registro completo de los mensajes del sistema de un *Smartphone*. En Android Studio existe el llamado Android Monitor, que es una pantalla donde se muestra el registro del Logcat y permite visualizar este registro.

Esto se utiliza a modo de comprobación de errores. Como se observa en la figura 3.32 se ha definido un String, estático y final, para que pueda ser accedido sin la necesidad de crear un objeto y no pueda modificarse, llamado, en este caso, *TAG*. A esta cadena de caracteres se le da un valor cualquiera, por lo general suele atender al nombre de la actividad en la que se encuentra. Como la pantalla del Logcat nos muestra todos los registros, a través del TAG, que en nuestro caso es *"actividad controlar"* se podrá realizar un filtro con estos caracteres, para que en el largo registro únicamente se muestren los que contienen esta cadena. De esta manera, como se indica en la figura 3.33, a través del Logcat sabremos si la aplicación ha pasado por esa línea de código. Esto es interesante cuando, programando, se da la situación de que la aplicación no se abre debido a un error en el código y no sabemos dónde se encuentra éste, o en nuestro caso, cuando se quiere enviar un dato al módulo Bluetooth y no podemos saber si lo ha realizado correctamente pasando por la función *write*.

Posteriormente a inicializar nuestra conexión con el módulo Bluetooth se da paso a declarar las funciones a realizar cuando se pulsan los botones. Los botones de velocidad son declarados con un OnClickListener, visto con anterioridad, y estos enviarán un '4#','5#' o '6#' dependiendo de la elección. El asterisco (#) se utiliza para detectar el final de línea, que será interpretado posteriormente por Arduino.

Para los controles de dirección, se ha decidido utilizar las funciones *setOnTouchListener()* y *onTouch()*. Estas se encargan de enviar un evento al realizar una acción, que en este caso, mediante *MotionEvent.ACTION_DOWN* y *MotionEvent.ACTION_UP* enviaremos dos parámetros a nuestro Arduino. Al presionar el botón se enviará un carácter, acompañado del fin de línea (#), y al soltar, otro diferente para indicar que el usuario ha dejado de pulsar el botón.

Para enviar toda esta información se utiliza la función *write()* del hilo encargado de la conexión, este hilo es el encargado de enviar y recibir datos, se muestra en la siguiente figura:

```
private class ConnectedThread extends Thread{
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket){
        OutputStream Envia = null;
        try{
            Envia=socket.getOutputStream();
        }catch (IOException e){}
        mmOutStream=Envia;
    }


    public void write (String input){
        try{
            mmOutStream.write(input.getBytes());
            Log.d(TAG, msg: "pasamos por el write con"+input);
        }catch(IOException e){
            Toast.makeText(getBaseContext(), text: "Error al realizar la escritura", Toast.LENGTH_LONG).show();
            finish();
        }
    }
}  //Thread
```

Figura 3.34 Hilo de conexión Bluetooth

A principio se declara un flujo de bytes de salida mediante un OutputStream, donde se almacenarán los datos que se quieran enviar y mediante la función write los enviaremos. Este envío se hace a través de un getBytes, cuya función es codificar el String, en este caso llamado *input*, y enviarlo como una secuencia de bytes, ya que así es como se comunica el Bluetooth. A continuación se analiza la recepción de datos de esta actividad en nuestro Arduino:

```
else if (opcion=='2'){
    while(salir==0){
            delay(20);
              String Dato="";
              while(Serial1.available()>0 && salir==0){
                  char c=Serial1.read();
                  while(c != '#'){
                    Dato=Dato+c;
                    c=Serial1.read();
                  }
                  Serial.println(Dato);
```

Figura 3.35 Recepción de datos para el control del vehículo

La variable *opcion* es una variable char, donde se almacena el número de la opción en la que estamos. En este apartado se extrae la cadena de caracteres enviada al módulo, de tal manera que si hay algún byte disponible en el Serial1 (*Serial.available()>0)*, este se va almacenando en un String hasta dar con el final de línea (#).

Con la función *read* conseguimos leer directamente los datos enviados en forma de carácter sin hacer una conversión de bytes. Tras obtener el dato enviado se hace una secuencia de else-if anidados para realizar la opción deseada.

En la siguiente figura se muestra como la librería AFMotor.h permite, de manera sencilla, mover los motores conectados a la motorShield, en este caso, hacia delante:

```
if(Dato.equals("0")){
    Setvel(vel,vel,vel,vel);
    Motor1.run (FORWARD);
    Motor2.run (FORWARD);
    Motor3.run (FORWARD);
    Motor4.run (FORWARD);
    Serial.println("Hacia arriba");
}
```

Figura 3.36 Movimiento de motores

Existen tres estados de movimiento en la librería; FORWARD, BACKWARD y RELEASE. El primer estado acciona los motores en un sentido, el segundo en sentido contrario y el release los mantiene quietos. Análogamente, en las opciones de girar y retroceder, se reciben los datos 1,2,3 y se accionan los motores en las direcciones correspondientes. La función *Setvel* es una función que recibe 4 parametros, en este caso un numero entero que puede ir desde el 0 al 255, siendo 0 velocidad nula o mínima y 255 velocidad máxima. Esta velocidad toma un valor en cuanto pulsamos los botones de velocidad baja, media o alta, adquiriendo el valor de 100, 150 y 230 respectivamente. En el momento en el que dejamos de pulsar cualquier boton de dirección, enviando el valor ‘10#’, el vehículo pasa a un estado de completo RELEASE.

De esta manera se consigue llevar un control preciso de los movimientos del vehículo.

3.3.3 Localización del vehículo

En esta opción nuestro *Smartphone* se encargará de obtener la distancia a nuestro módulo Bluetooth y mostrarla en el mapa a través de nuestra ubicación junto con un círculo que indicará la distancia al módulo.

Para ello se ha creado una nueva actividad, pero, esta vez no se trata de una activity común, sino de una *Google Maps Activity*, que permite introducir la interfaz de Google Maps para dicha actividad.

Para utilizar los servicios de Google ha sido necesario habilitar ciertas API para nuestro proyecto desde internet, en Google Cloud Console. Algunas de las Api habilitadas han sido *Geolocation Api*, para utilizar la información de nuestra ubicación o *Maps SDK for Android*, que permite utilizar mapas basados en Google Maps y sus servidores, *Geocoding Api*, para las coordenadas…etc.

En esta actividad se dispondrá de una interfaz que mostrará nuestra ubicación en el mapa y un botón para actualizar el nivel de señal si nos hemos movido de nuestra posición.

Para mostrar el mapa es necesaria la función *onMapReady*, mostrada en la figura 3.37.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
        ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED)
        return;
    }
    mMap.setMyLocationEnabled(true);

    Context context=this;
    LocationManager locationManager=(LocationManager) context.getSystemService(Context.LOCATION_SERVICE);
    locationManager.requestLocationUpdates( LocationManager.GPS_PROVIDER,
            minTime: 200,
            minDistance: 1, locationListener);
}
```

Figura 3.37 Función onMapReady de la actividad MapsActivity

En primer lugar se declara el objeto mMap de la clase GoogleMap y se dan permisos para utilizar la ubicación, posteriormente se muestra en dicho mapa nuestra ubicación y para acceder a sus servicios y aplicaciones se declara un

LocationManager, que se utilizará cada vez que nuestra posición cambie de ubicación con la función *onLocationChanged*. En dicha función se obtendrá de nuevo los valores de Latitud y Longitud de nuestra ubicación y se borrarán los elementos dibujados en el mapa.

Con esta configuración se muestra el mapa de la figura 3.38. En dicho mapa se encuentra expuesta la ubicación del Smartphone y un botón de actualizar.

Este botón sirve para realizar la búsqueda del módulo Bluetooth y encontrar la distancia que nos separa para posteriormente dibujar un circulo que rodee a nuestra ubicación con los valores recibidos, simulando de esta manera el área en el que se encuentra dicho módulo.

Figura 3.38 Interfaz gráfica de la actividad MapsActivity

Para encontrar la distancia al módulo se utiliza el dato RSSI (*Received Signal Strength Indicator* o Indicador de fuerza de la señal recibida), medido en dBm, que utiliza un rango de 0 a -100 dBm siendo 0 la señal ideal pero difícil de

conseguir en la práctica y -100 la señal mínima que conlleva la pérdida de datos. Para conseguir esta señal se ejecuta la función *startDiscovery* cada vez que pulsamos el botón de actualizar, donde, anteriormente se ha registrado un BroadcastReceiver, que se ejecutará cuando el adaptador realice una acción, en este caso, startDiscovery. En la siguiente figura se muestra cómo:

```
registerReceiver(receiver, new IntentFilter(BluetoothDevice.ACTION_FOUND));
botonactualiz.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        btAdapter.startDiscovery();
    }
});
```

Figura 3.39 Llamada al BroadcastReceiver desde MapsActivity

El startDiscovery se encarga de encontrar los dispositivos Bluetooth que se encuentran al alcance del Smartphone. Cuando el Smartphone detecta, en este caso, nuestro módulo Bluetooth llamado "VIBT" (Vehículo Inteligente Bluetooth) obtenemos el dato *EXTRA_RSSI* del dispositivo.

Para conseguir la distancia a partir del dato RSSI, se ha realizado una serie de pruebas donde se ha medido dicho dato en función de la distancia, a través de un medidor láser de distancias.

En la siguiente figura se muestra una gráfica con el resultado de las pruebas realizadas, siendo el eje Y la señal RSSI y el eje X la distancia medida en metros.

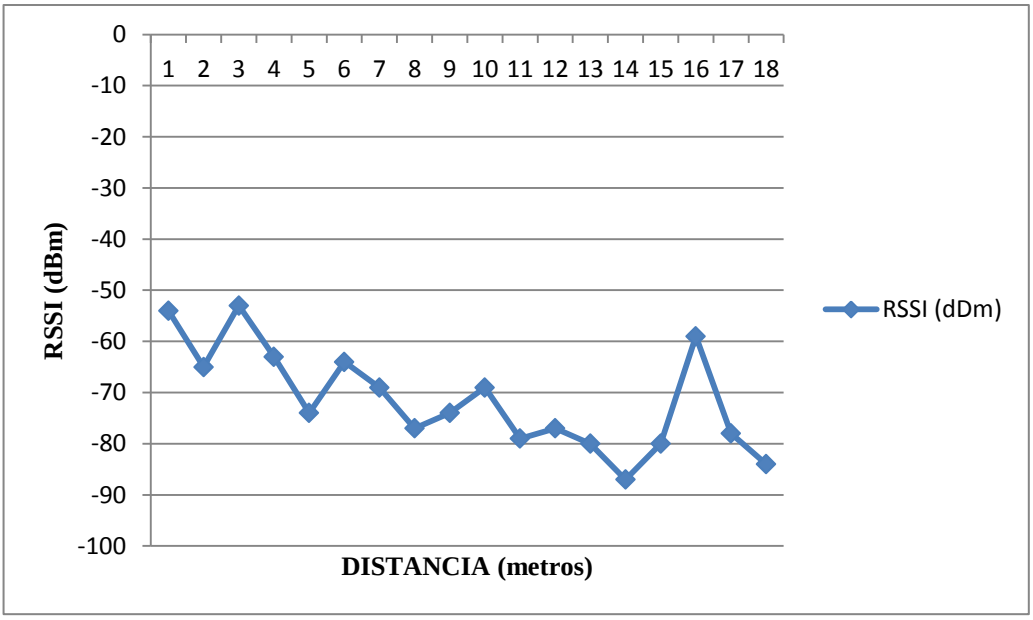


Figura 3.40 Señal RSSI en función de la distancia

La señal RSSI, como se observa, no es muy precisa, ya que depende de las características del módulo, de las características del ambiente, del posible ruido interno o externo.etc., pero si se puede observar cómo, cuanto más nos alejamos del módulo, menor son los dBm que recibimos.

Para realizar una conversión adecuada para el módulo VIBT, nos hemos basado en los datos anteriores, por lo que, en la aplicación, se ha establecido un rango de valores a través de else-if anidados donde, según los dBm recibidos, nos encontramos a 1, 3, 5, 8, 12 o más de 15 metros. Una vez obtenida la distancia aproximada, se almacena dicho valor en una variable entera para poder dibujar un círculo en el mapa.

En la siguiente figura se muestra como crear un círculo en una actividad de Google Maps:

```
mMap.clear();
Circle circle = mMap.addCircle(new CircleOptions()
        .center(new LatLng(Latitud,Longitud))
        .radius(radio) //En metros
        .strokeColor(Color.RED));
btAdapter.cancelDiscovery();
```

Figura 3.41 Crear círculo en una Google Map Activity

Previamente se elimina el círculo anterior (si lo hubiera) y toda figura disponible en el mapa para dar pie al siguiente círculo. Este tendrá su centro en el punto de nuestra ubicación, que habrá tomado los valores previos de Latitud y Longitud, y un radio del tamaño indicado en los else-if anteriores, expresado en metros. Una vez realizado esto, indicamos al Smartphone que deje de buscar dispositivos para no perder tiempo en la aplicación. En la siguiente figura se muestran dos ejemplos de esta actividad y en ellas, un TextView para mostrar la distancia en metros y la señal RSSI obtenida, en dBm:

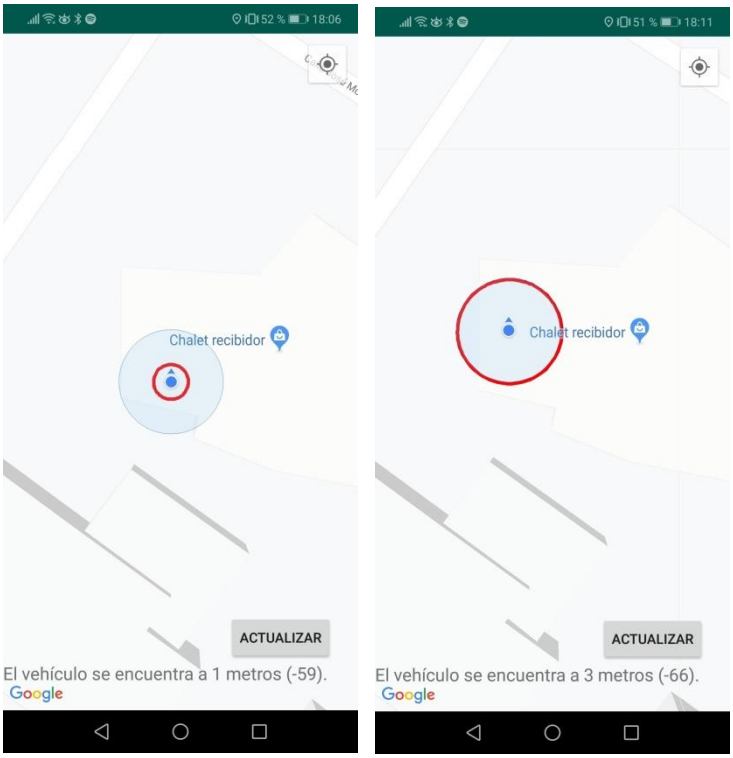


Figura 3.42 Interfaz gráfica de la actividad MapsActivity

CAPÍTULO 4. CONCLUSIÓN

Tras la realización de este proyecto, se puede concluir que se han cumplido la mayoría de objetivos propuestos en un principio.

Se ha conseguido desarrollar una aplicación sólida, sencilla y clara, donde cualquier tipo de usuario puede hacer uso de ella. La aplicación nos ha permitido realizar un acercamiento mucho más profundo al sistema operativo Android, descubriendo sus funciones y posibilidades, y su herramienta Android Studio, donde se ha realizado la aplicación, permitiendo conocer su funcionamiento más a fondo. También nos ha permitido conocer y utilizar las API en nuestro proyecto, siendo estas un concepto que extiende en gran medida las posibilidades de una aplicación.

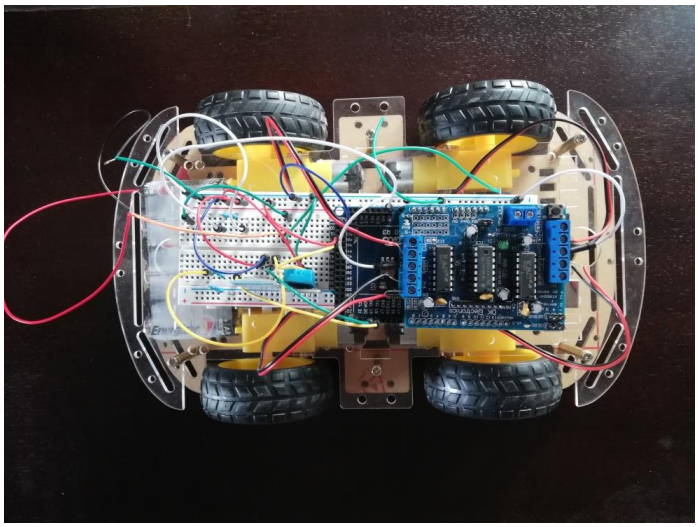
Al utilizar estas API, el objetivo de conseguir una aplicación que no necesite ser administrada por ninguna entidad se ve afectado, ya que, al hacer uso de las tecnologías de Google y sus servicios nos exponemos a una dependencia de estos, a pesar de que su uso sea positivo para la aplicación.

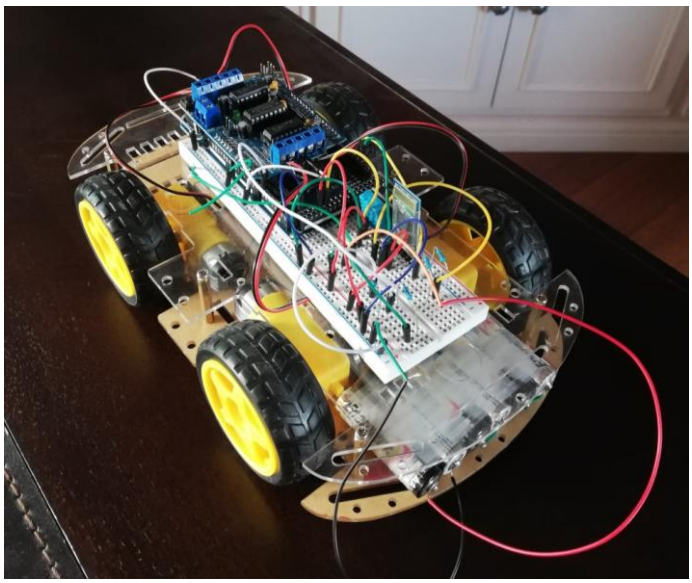
Otro de los objetivos cumplido ha sido el acercamiento a las tecnologías de comunicación. Se ha trabajado con la tecnología Bluetooth y se ha comprendido como funciona y como interactúa con un Smartphone a través del envío y recepción de datos. La conexión entre ambos dispositivos ha permitido también acercarnos al concepto del Internet de las Cosas y cómo hacer para comunicarse entre dos objetos de manera remota, en este caso, entre Arduino y Android, facilitando así las tareas al ser humano.

Como aspecto negativo del proyecto cabe destacar la utilización de la tecnología Bluetooth a la hora de localizar el vehículo. Al no tratarse de una tecnología que puede conectarse con Internet como el Wi-Fi, la forma de obtener la ubicación del vehículo es algo rudimentaria, ya que, si dispusiéramos de una ubicación GPS del módulo, sería tan sencillo como mostrarla en el mapa junto con nuestra ubicación. Al disponer de tecnología Bluetooth la forma más sencilla para obtener su ubicación ha sido a través de la combinación entre la señal RSSI y nuestra ubicación a través de un área de distancia.

Otro de los aspectos negativos a destacar ha sido la incapacidad de mantener la conexión al cambiar de actividad. Lo óptimo hubiera sido no desconectarnos del módulo al cambiar entre las distintas opciones del menú principal. Por falta de tiempo no ha podido solucionarse este aspecto y se ha optado por realizar la conexión de nuevo al cambiar de actividad.

A continuación se muestra el prototipo a pequeña escala que se ha realizado para el desarrollo del Trabajo de Fin de Grado y su aplicación móvil.





LISTA DE ABREVIATURAS

ADT (*Android Development Tools*, Herramientas de desarrollo Android).

AI (Adobe Illustrator).

AOT (*Ahead of Time*, Compilación anticipada).

API (*Application Programming Interface*, Interfaz de Programación de Aplicaciones).

App (*Application*, Aplicación móvil).

ART (*Android Runtime*, Tiempo de ejecución de Android).

BLE (*Bluetooth Low Energy*, Bluetooth de baja energía).

Bps (Bits por segundo).

dBm (decibelios con referencia a milivatios).

GUI (*Graphical User Interface*, Interfaz gráfica de usuario).

HS (*High-Speed*, Alta Velocidad).

IBM (*International Business Machines Corp*.).

ICSP (*In Chip Serial Programmer*, Programación serial en circuito).

Id (*Identification*, Identificación).

IDE (*Integrated Development Enviroment*, Entorno de Desarrollo Integrado).

Inc (*Incorporation*, Corporación).

IoT (*Internet of things*, Internet de las Cosas).

JIT (*Just-in-time*, Justo a tiempo).

kbps y Mbps (Kilobits por segundo y Megabits por segundo).

LAN (*Local Area Network*, Red de Área Local).

MAC (*Media Access Control*, Control de Acceso al Medio).

MB y GB (Megabytes y Gigabytes).

mV (Milivatios).

PCB (*Printed Circuit Board*, Placa de circuito impreso).

RSSI (*Received Signal Strength Indicator*, Indicador de Fuerza de la Señal Recibida).

S.O. (Sistema operativo).

SBC (*Single Board Computer*, Ordenador de Placa Reducida).

UART's (*Universal Asynchronous Receiver-Transmitter*, Transmisor-Receptor Asíncrono Universal).

UUID (*Universal Unique Identifier*, Identificador Único Universal).

VIBT (Vehículo Inteligente Bluetooth).

WECA (*Wireless Ethernet Compatibility Alliance*, Alianza de Compatibilidad Ethernet Inalámbrica).

Wi-Fi (*Wireless Fidelity*, Fidelidad Inalámbrica).

Xml (*Extensible Markup Language*, Lenguaje de Marcado Extensible).

BIBLIOGRAFÍA

[1] Análisis del internet de las cosas [Ilustración]. (s.f.). Recuperado 7 junio, 2019, de https://www.grandviewresearch.com/industry-analysis/internet-of-things-iot-analytics-market

[2] Nicolás, M. (2017, 11 enero). ¿Qué es retail? Definición y características. Recuperado 7 junio, 2019, de https://www.oleoshop.com/blog/que-es-retail

[3] Lued, K. L. (2015, 2 febrero). Las 10 aplicaciones de Internet de las cosas más populares en este momento [Ilustración]. Recuperado 7 junio, 2019, de https://iot-analytics.com/10-internet-of-things-applications/

[4] NTS [Ilustración]. (s.f.). Recuperado 15 junio, 2019, de https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/729524/nts-factsheets.pdf

[5] Seguridad vial. (s.f.). Recuperado 16 junio, 2019, de https://www.race.es/documentos/seguridad_vial/formacion_vial/Informe%20RACE%20BP%20CASTROL%20Uso%20de%20los%20smartphones%20en%20la%20conduccion.pdf

[6] Vehículo autónomo. (2019, 14 de marzo). *Wikipedia, La enciclopedia libre*. Fecha de consulta: Mayo 16, 2019 desde https://es.wikipedia.org/w/index.php?title=Veh%C3%ADculo_aut%C3%B3nomo&oldid=114579665.

[7] Self-Drive Cars and You: A History Longer than You Think. (2014, 5 agosto) [Ilustración]. Recuperado 16 mayo, 2019, de https://www.velocetoday.com/self-drive-cars-and-you-a-history-longer-than-you-think/

[8] Distribution dashboard | Android Developers. (s.f.). Recuperado 8 mayo, 2019, de: https://developer.android.com/about/dashboards

[9] Sistemas operativos. (s.f.). Recuperado 9 junio, 2019, de https://www.areatecnologia.com/sistemas-operativos.htm

[10] Mobile Operating System Market Share Worldwide [Ilustración]. (s.f.). Recuperado 9 junio, 2019, de http://gs.statcounter.com/os-market-share/mobile/worldwide/#yearly-2009-2019

[11] Rodriguez, G. (2012, 10 octubre). Symbian, el Sistema Operativo móvil. Recuperado 9 junio, 2019, de https://www.vix.com/es/btg/tech/2007/03/12/symbian-el-sistema-operativo-movil

[12] Acerca de Tizen. (s.f.). Recuperado 9 junio, 2019, de https://www.tizen.org/about

[13] Windows Mobile. (s.f.). Recuperado 9 junio, 2019, de https://es.wikipedia.org/wiki/Microsoft_Mobile

[14] Blackberry. (s.f.). Recuperado 9 junio, 2019, de https://es.wikipedia.org/wiki/BlackBerry

[15] Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2018 [Ilustración]. (s.f.). Recuperado 9 junio, 2019, de https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

[16] Rus, C. (2016, 13 octubre). La evolución de iOS desde sus orígenes: una carrera para ser el mejor sistema operativo móvil de la historia. Recuperado 9 junio, 2019, de https://www.applesfera.com/ios/la-evolucion-de-ios-desde-sus-origenes-una-carrera-para-ser-el-mejor-sistema-operativo-movil-de-la-historia

[17] Lopez, Jose María. (2018, 27 agosto). De la A a la P: 10 años de historia de Android. Recuperado 16 mayo, 2019, de https://blogthinkbig.com/diez-anos-de-historia-de-android

[18] Arquitectura de la plataforma. (s.f.). Recuperado 10 junio, 2019, de https://developer.android.com/guide/platform?hl=es-419

[19] Android arquitectura [Ilustración]. (s.f.). Recuperado 12 junio, 2019, de https://sites.google.com/site/swcuc3m/home/android/generalidades/2-2-arquitectura-de-android

[20] Android. (s.f.) [Ilustración]. Recuperado 10 junio, 2019, de https://es.wikipedia.org/wiki/Android

[21] Eclipse (software). (s.f.). Recuperado 12 junio, 2019, de https://es.wikipedia.org/wiki/Eclipse_(software)

[22] ¿Que es Xamarin? (2018, 25 marzo). Recuperado 12 junio, 2019, de https://okdiario.com/tecnologia/que-xamarin-2022974

[23] [Logo Xamarin] [Ilustración]. (s.f.). Recuperado 12 junio, 2019, de https://commons.wikimedia.org/wiki/File:Xamarin_logo_and_wordmark.png

[24] Quintero, J. (2016, 2 junio). Android Studio vs Appcelerator. Recuperado 11 junio, 2019, de https://prezi.com/93ln3oviuw0v/android-studio-vs-appcelerator/

[25] Logo Appcelerator [Ilustración]. (s.f.). Recuperado 12 junio, 2019, de https://www.appcelerator.com/legal/trademark-policy/

[26] Caules, C. A. (2015, 1 abril). ¿Que es Gradle? [Ilustración]. Recuperado 11 junio, 2019, de https://www.arquitecturajava.com/que-es-gradle/

[27] [Android Studio]. (s.f.). Recuperado 12 junio, 2019, de https://es.wikipedia.org/wiki/Android_Studio

[28] Ramírez, C. G. (s.f.). Diseño y análisis de comunicaciones en Android [TFG]. Recuperado 12 junio, 2019, de http://oa.upm.es/49714/1/TFG_CARMEN_GIOVANETTI_RAMIREZ.pdf

[29] [Tipos de comunicacion] [Ilustración]. (s.f.). Recuperado 11 junio, 2019, de http://veraview.com/the-different-forms-of-streaming/

[30] Network topologies [Ilustración]. (s.f.). Recuperado 11 junio, 2019, de https://es.wikipedia.org/wiki/Topolog%C3%ADa_de_red#/media/Archivo:NetworkTopologies.svg

[31] Bluetooth: Que es? Para qué sirve? Como usar Bluetooth? Problemas de conexión. (s.f.). Recuperado 6 junio, 2019, de https://tecnologia-informatica.com/Bluetooth/

[32] [Definición maestro-esclavo]. (s.f.). Recuperado 12 junio, 2019, de http://diccionario.raing.es/es/lema/maestro-esclavo

[33] Wireless-Communication-Bluetooth [Ilustración]. (s.f.). Recuperado 12 junio, 2019, de https://www.tutorialspoint.com/wireless_communication/wireless_communication_bluetooth.htm

[34] Formato de paquetes Bluetooth. (s.f.). Recuperado 12 junio, 2019, de https://www.electronicafacil.net/tutoriales/Formato-paquetes-Bluetooth.html

[35] Garreta, J. (2017, 13 enero). Historia y evolución de las redes Wi-Fi. Recuperado 12 junio, 2019, de http://pleasenetworks.com/blog/post/13/historia-y-evolucin-de-las-redes-wifi-de-tecnologa-inalmbrica-a-aplicacin-ligera-para-retailers

[36] Diseño abierto. (s.f.). Recuperado 12 junio, 2019, de https://es.wikipedia.org/wiki/Dise%C3%B1o_abierto

[37] Arduino: Tecnología para todos. (s.f.). Recuperado 16 mayo, 2019, de https://arduinodhtics.weebly.com/historia.html

[38] Diosdado, R. (s.f.). Conociendo Arduino [Ilustración]. Recuperado 12 junio, 2019, de https://www.zonamaker.com/arduino/intro-arduino/conociendo-arduino-introduccion

[39] Raspberry PI. (2013, 18 diciembre). Recuperado 12 junio, 2019, de https://histinf.blogs.upv.es/2013/12/18/raspberry-pi/

[40] ARDUINO MEGA 2560. (s.f.). Recuperado 17 mayo, 2019, de http://arduino.cl/arduino-mega-2560/

[41] Baudios y bits. (s.f.). Recuperado 20 mayo, 2019, de https://www.monografias.com/docs/Baudios-y-bits-hay-alguna-relacion-P3JJVMZZMY

[42] [Perfil TECNOIOT]. (s.f.). Recuperado 21 mayo, 2019, de https://www.amazon.es/TECNOIOT/s?k=TECNOIOT

[43] [Instalación Fritzing - 64 bits] [Programa]. (s.f.). Recuperado 21 mayo, 2019, de http://fritzing.org/download/

[44] Currey, M. (2014, 28 octubre). Arduino HC-05 [Ilustración]. Recuperado 20 mayo, 2019, de http://www.martyncurrey.com/arduino-with-hc-05-Bluetooth-module-at-mode/

[45] Sensor de temperatura y humedad Dht11 [Fotografía]. (s.f.). Recuperado 5 junio, 2019, de https://articulo.mercadolibre.com.mx/MLM-553295702-sensor-de-temperatura-y-humedad-dht11-para-arduino-pic-_JM?quantity=1

[46] Driver Shield [Ilustración]. (s.f.). Recuperado 14 junio, 2019, de https://naylampmechatronics.com/arduino-shields/86-shield-motor-driver-l293d.html

[47] DHT11 Library. (s.f.). Recuperado 5 junio, 2019, de https://learn.adafruit.com/dht/downloads

[48] ICRA. (2015, 18 marzo). Ciclo de vida de una *Activity* [Ilustración]. Recuperado 18 mayo, 2019, de http://icra.blogspot.es/1426639560/ciclo-de-vida-de-una-activity/